SPORE

Spore:
A Massively *Single-Player* Online Game

Andrew Willmott, Lead Engineer

Maxis

Electronic Arts

Maxis

- SimCity, Sims, <cough>Sims Online<cough>
- Ongoing trend: player-originated content
  - Started with mods, new textures, BAT
  - Went to another level with Sims: houses, objects, story-telling, extensive sim modification
  - Continued with Sims 2: see http://thesims2.ea.com/exchange/
- Encouraging player content a big deal for us

SPORE

Spore

- Elevator pitch
  - Play from a cell to a galaxy-spanning civilization

- But, behind-the-scenes agenda:
- Take player-created content to the next level
  - Assume that *everything* is generated/modifiable
- Build content-sharing in to the game
  - So you are automatically playing with other player's content

MMOs

- Massively Multiplayer Online Game
- Persistent World
- Broadcast avatar location and world state
- Hundreds or thousands of players simultaneously

MSO

- Massively Single-player Online Game
- Substitute game world with *Content*
- Players create parts of their own world
- Content is pollinated to/from other players
  - You create your own avatar and associated assets, everything else is from other players
  - For non-network play, ship with a set of Maxis assets
- Content from tens of thousands of players

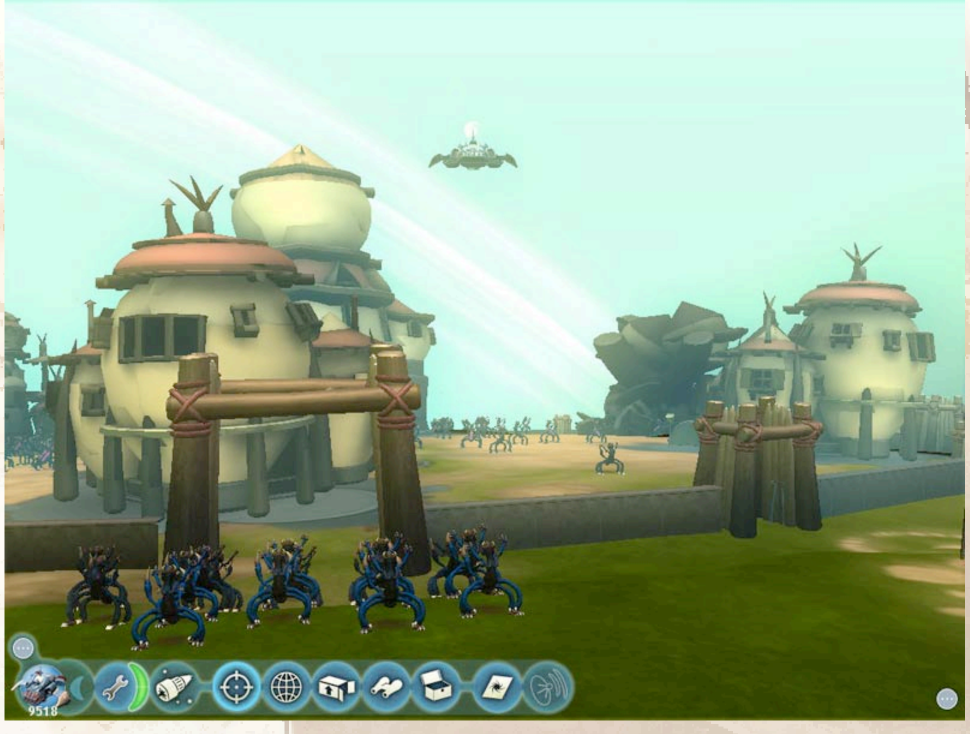
Spore: Player-Created Stuff

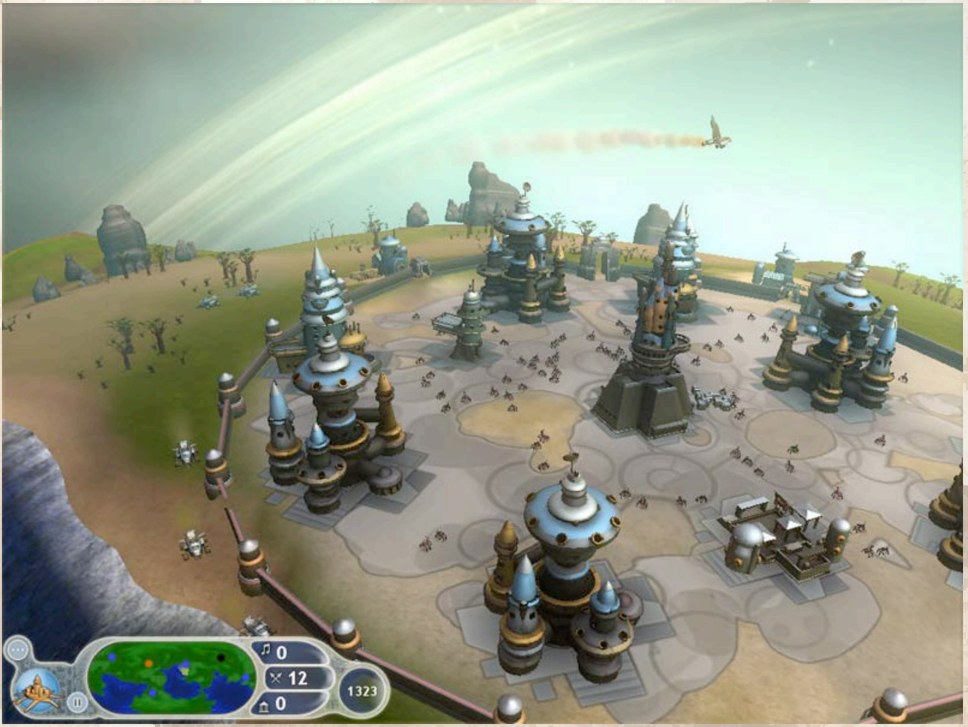
- Want players to be able to create key parts of their game
- Pollinate player-created things via servers, so your game is made of both your own creations and others’
- Richer experience, less art work(!)

SPORE

Spore

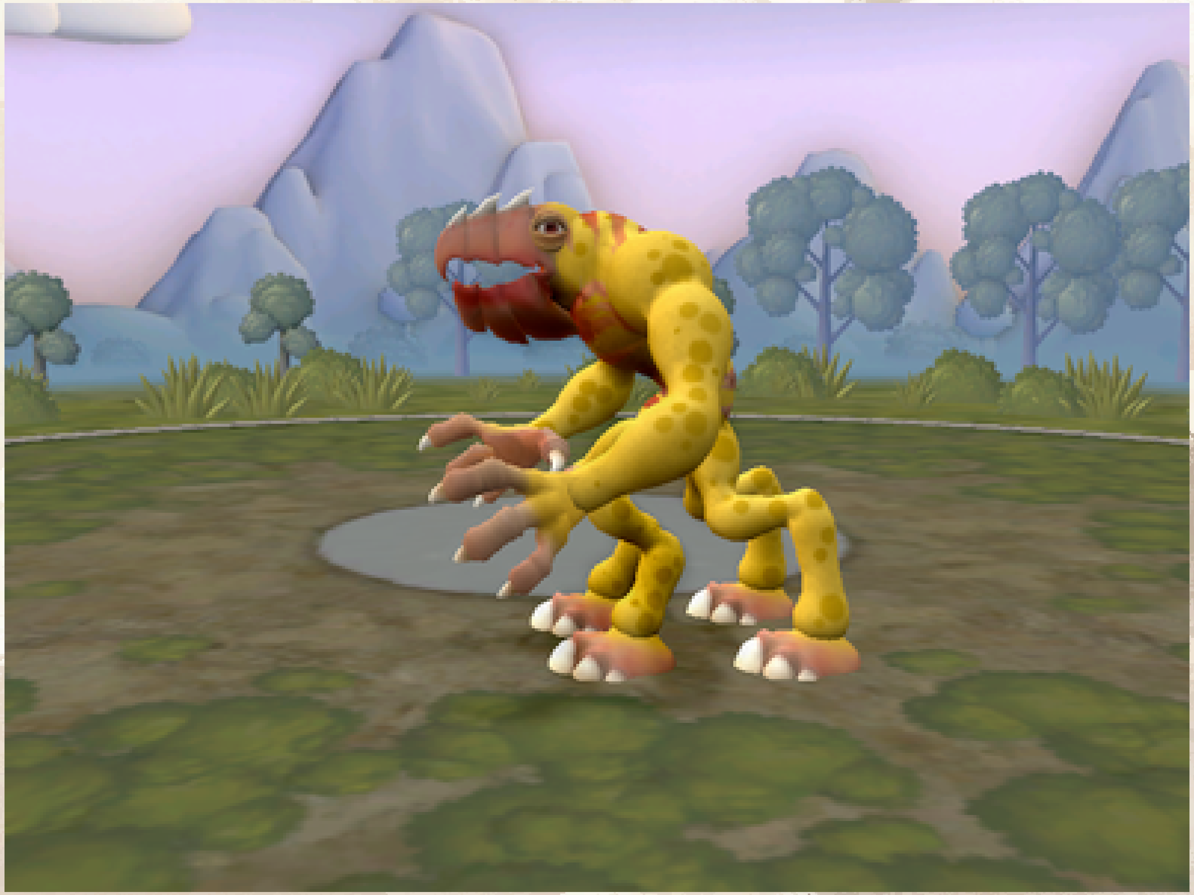
- Creatures, Buildings, Vehicles...





SPORE

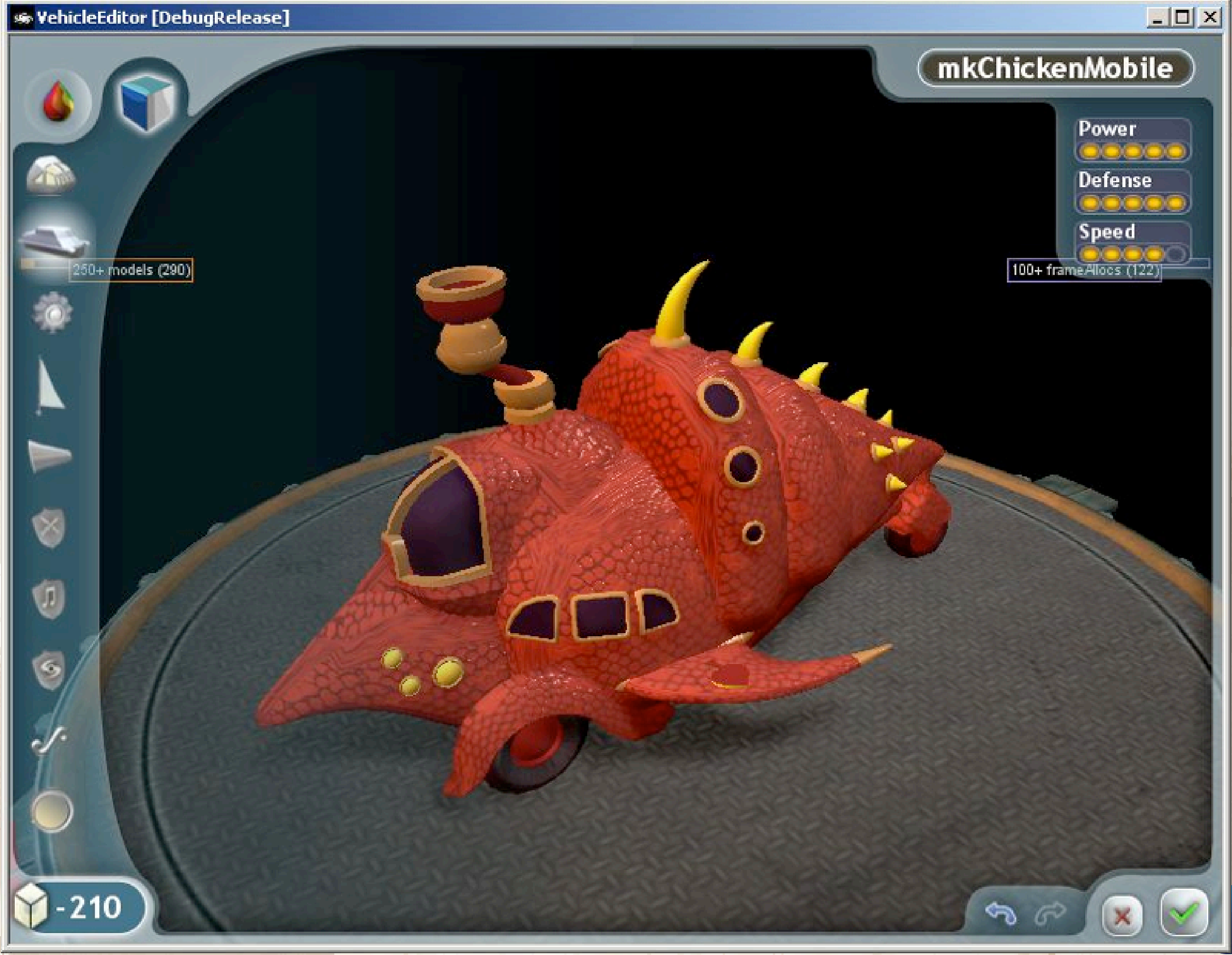
Player-Created Stuff: Creatures



Player-Created Stuff: Buildings

SPORE

Player-Created Stuff: Cars



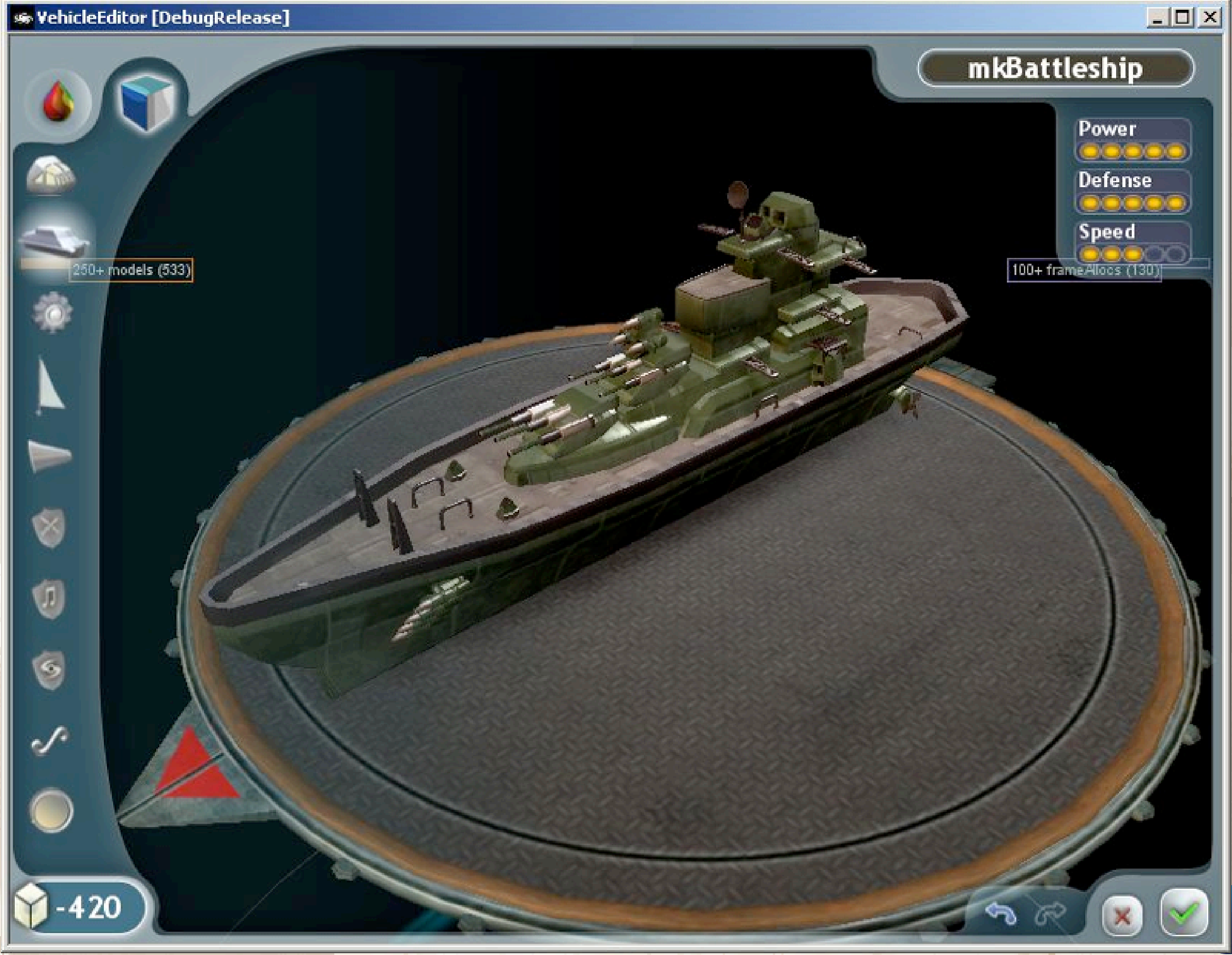
VehicleEditor [DebugRelease]

mkChickenMobile

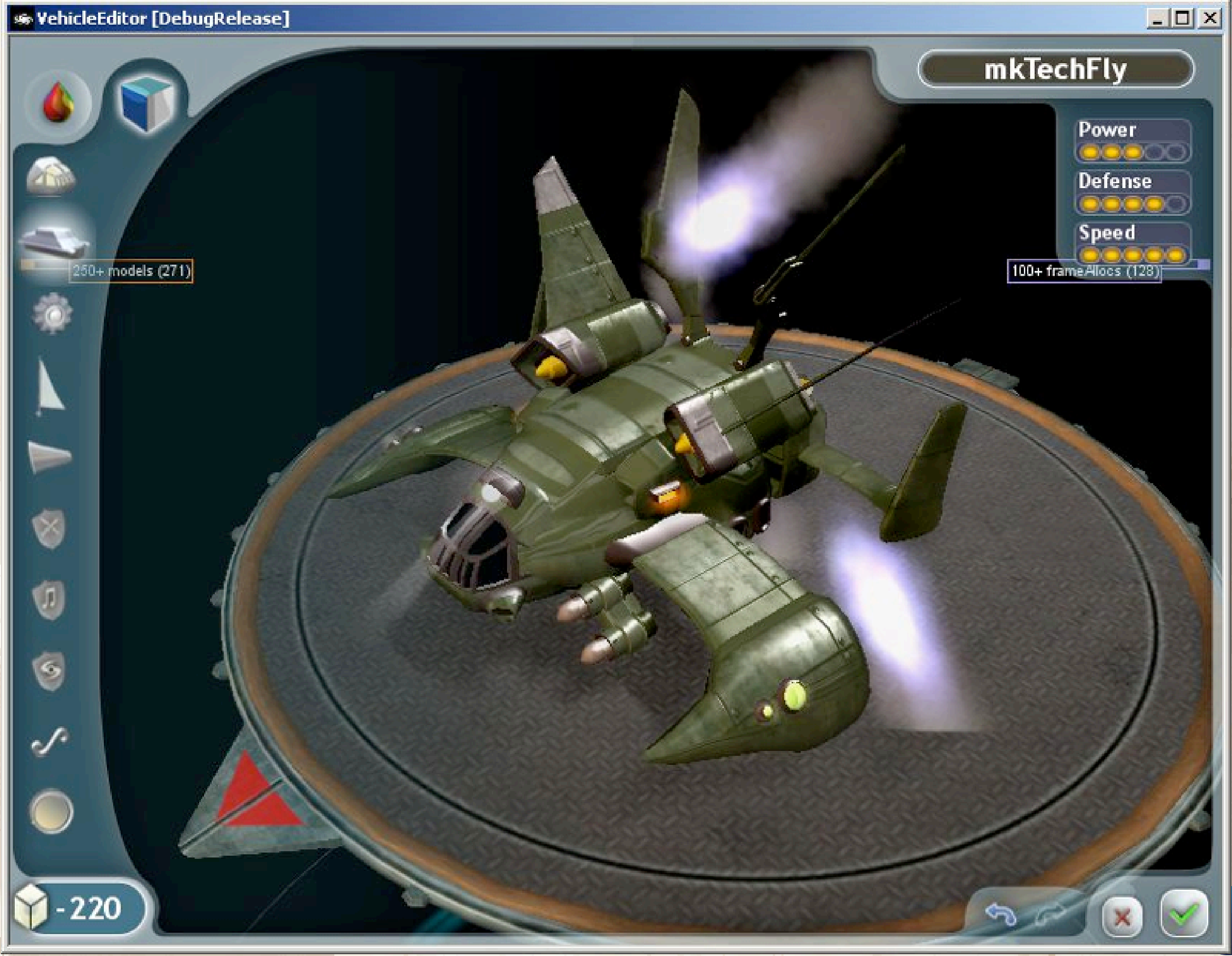
250+ models (290)

100+ frameAllocs (122)

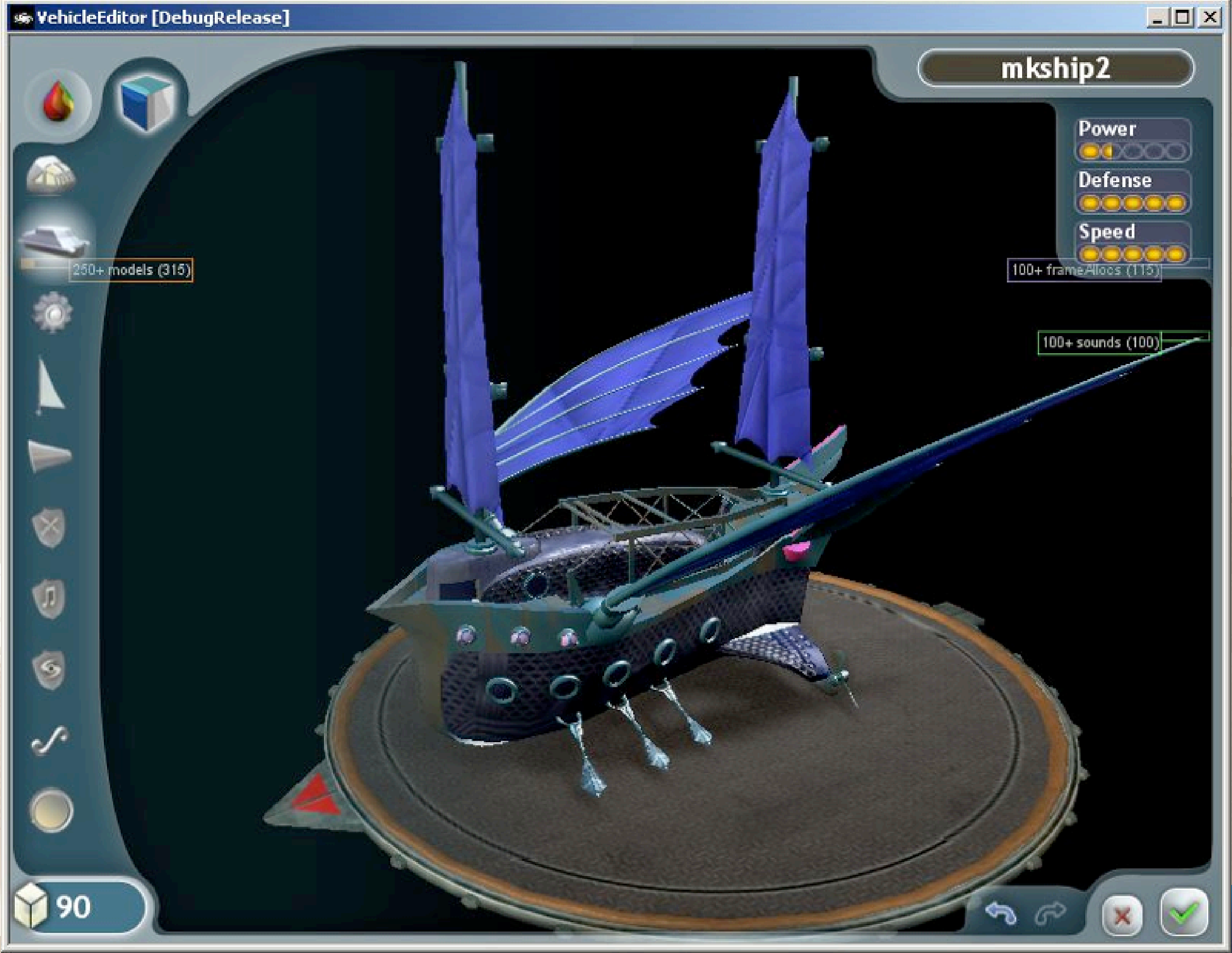
Player-Created Stuff: Boats



Player-Created Stuff: Planes



Player-Created Stuff: Hybrids



Why not MMO too?

- Game is single-player: reduce risk (hollow laugh)
- Goals
  - Story-telling
  - Richer World
  - Sense of scale
- Didn't need multiplayer support to achieve this

SPORE

Spore Development

- <mumble> - 2005: concept development
- GDC 2005, initial announcement, showed prototype with pieces knitted together in Sims 2 engine.
- June 2005, started on Spore engine & associated games
- Shipping some time in 2008

Technology

- Player Model Creation
- Player Model Texturing
- Animation: procedural, deformation, runtime, skeleton re-targeting...
- Procedural Planet generation
- Spherical Planets
- Editor Interaction
- Procedural & Player-generated Audio

SPORE

Technology

- Pollination: aesthetic matching, queries, tags
- Five individual games:
  - – pathing, collision detection, procedural level layout...
- Galaxy representation
- Engine stuff: lighting, effects system, material system, resource control, job manager, ...
- Steganography (MiP: Model-in-Picture)
- Debug web server
- ...

Technology: Today

- Player Model Creation
- Player Model Texturing
- Animation: procedural, deformation, runtime, skeleton re-targeting...
- Procedural Planet Generation
- Spherical Planets
- Editor Interaction
- Procedural & player-generated audio

Rigblocks: Player-Deformable Objects

SPORE

Lydia Choy, Ryan Ingram, Ocean Quigley,
Brian Sharp, Andrew Willmott

How can players create models?

- Let player use supplied parts to build model
  - Allow stacking, pinning, sliding
- *But*, static is boring, requires many blocks to be expressive. So
  - Add animations that *deform* blocks
  - Animations driven by player-controlled handles
- Result: Rigblocks, our LEGO$_{\text{(tm)(R)(whatever)}}$

SPORE

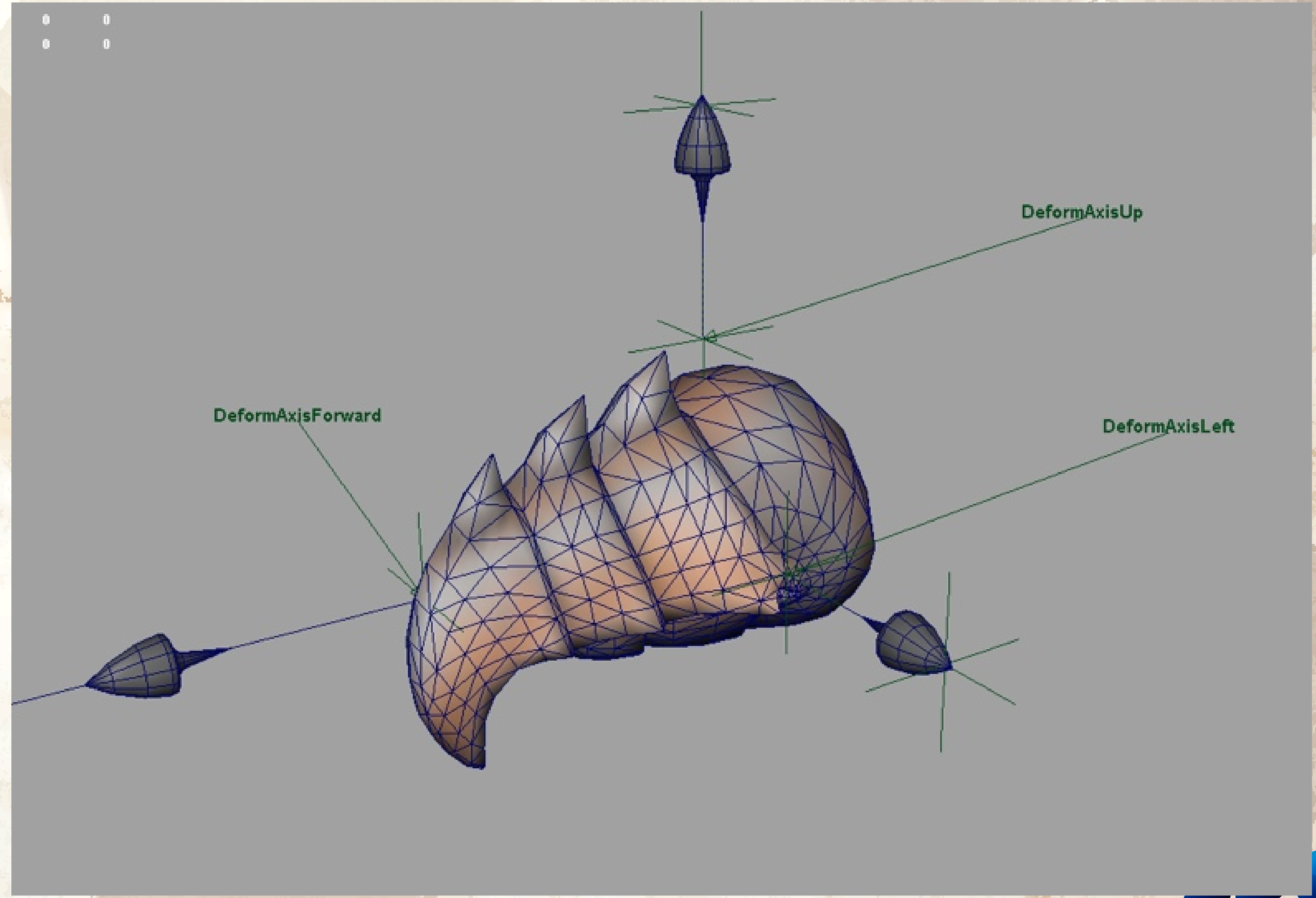
Advantages

- Player interaction with the block is intuitive and straightforward
- Rigblock deformations are expressive
- Provides a balance between enabling player creativity and amplifying player creativity

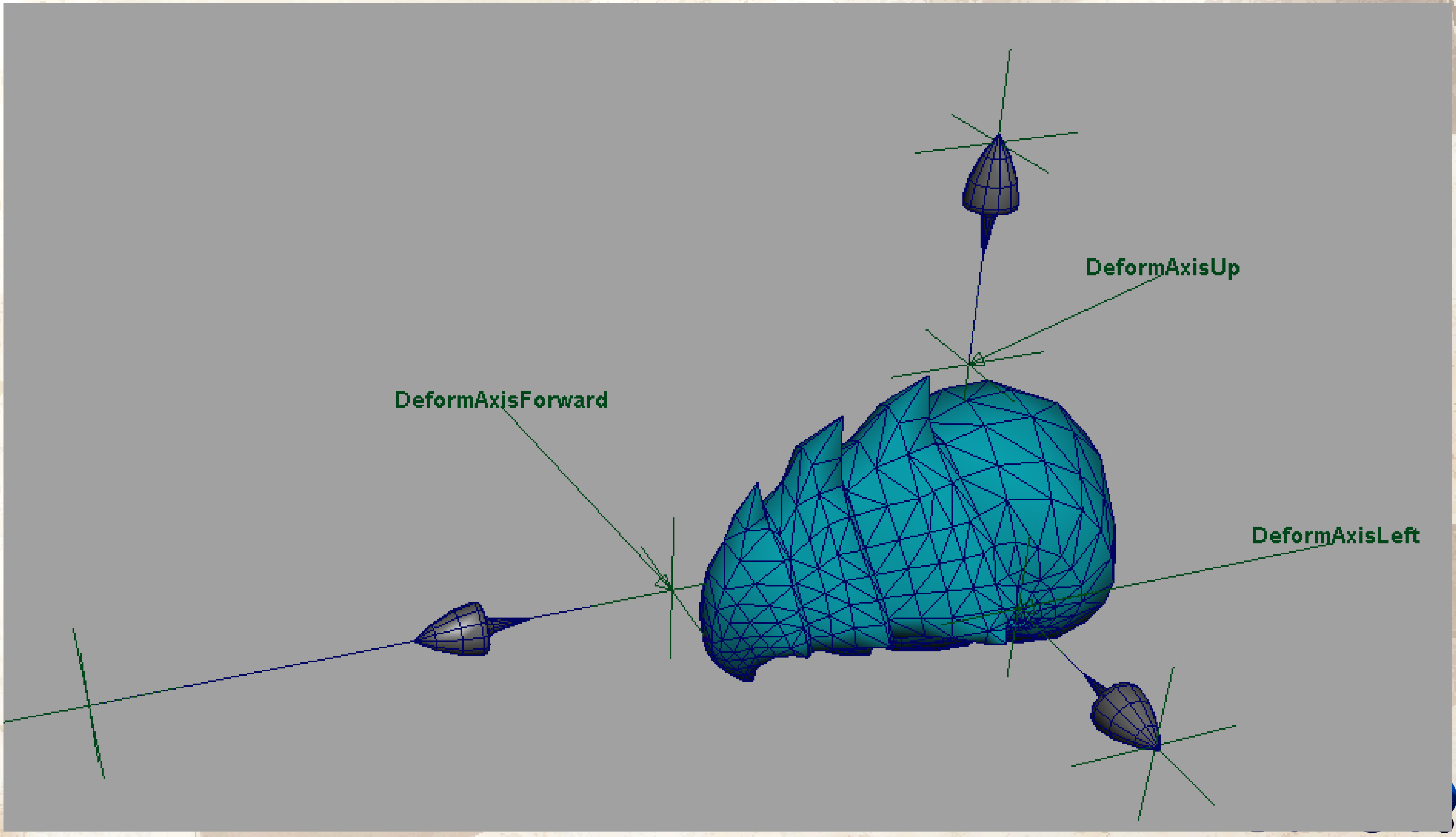
Advantages

- Aiming for the sweet spot between:
  - High-quality, artist-created models, with no player control
  - Lower-quality, effort-intensive, wholly player-driven approach, such as providing a sculpting tool.

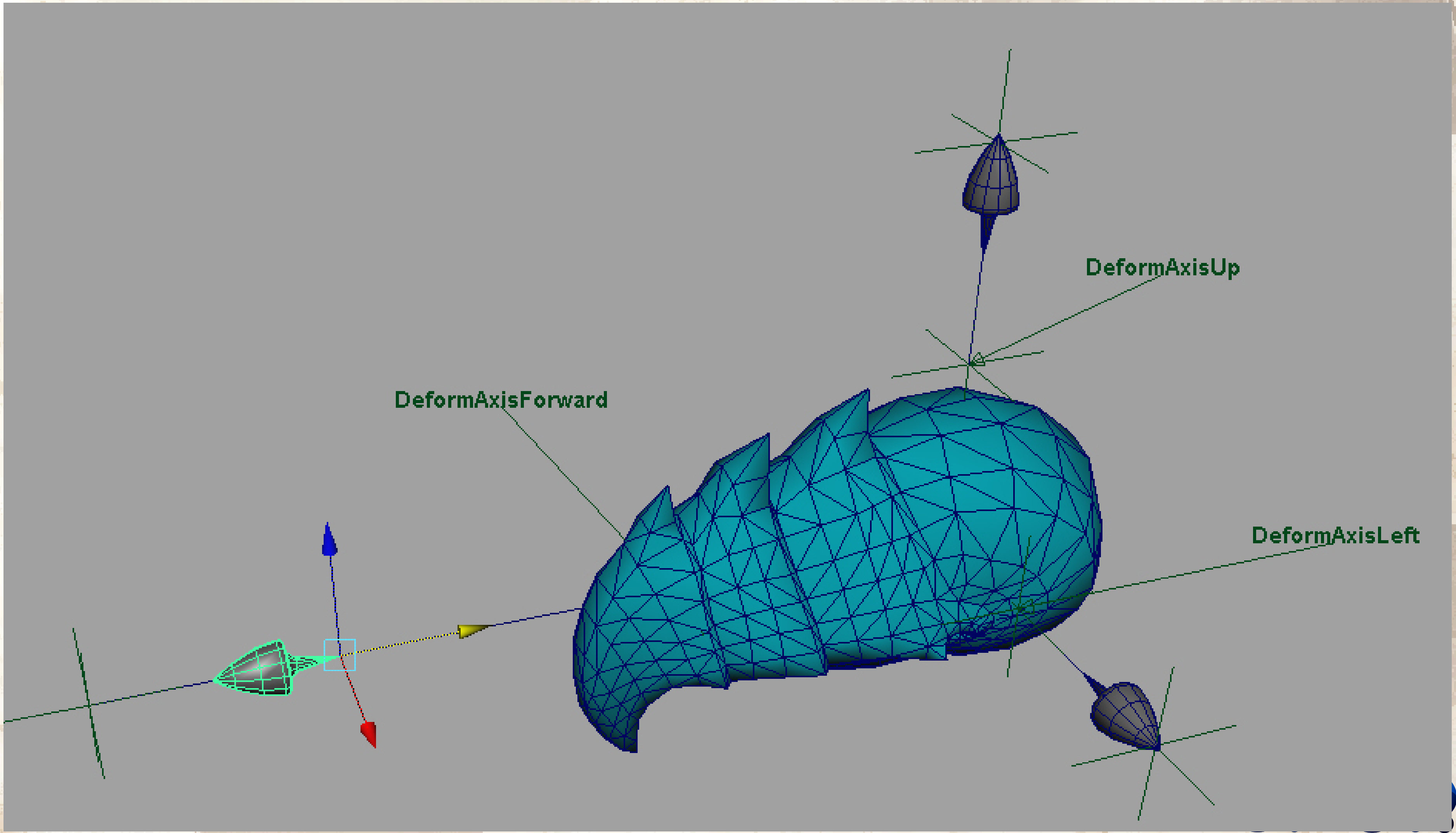
Example: Maya Model



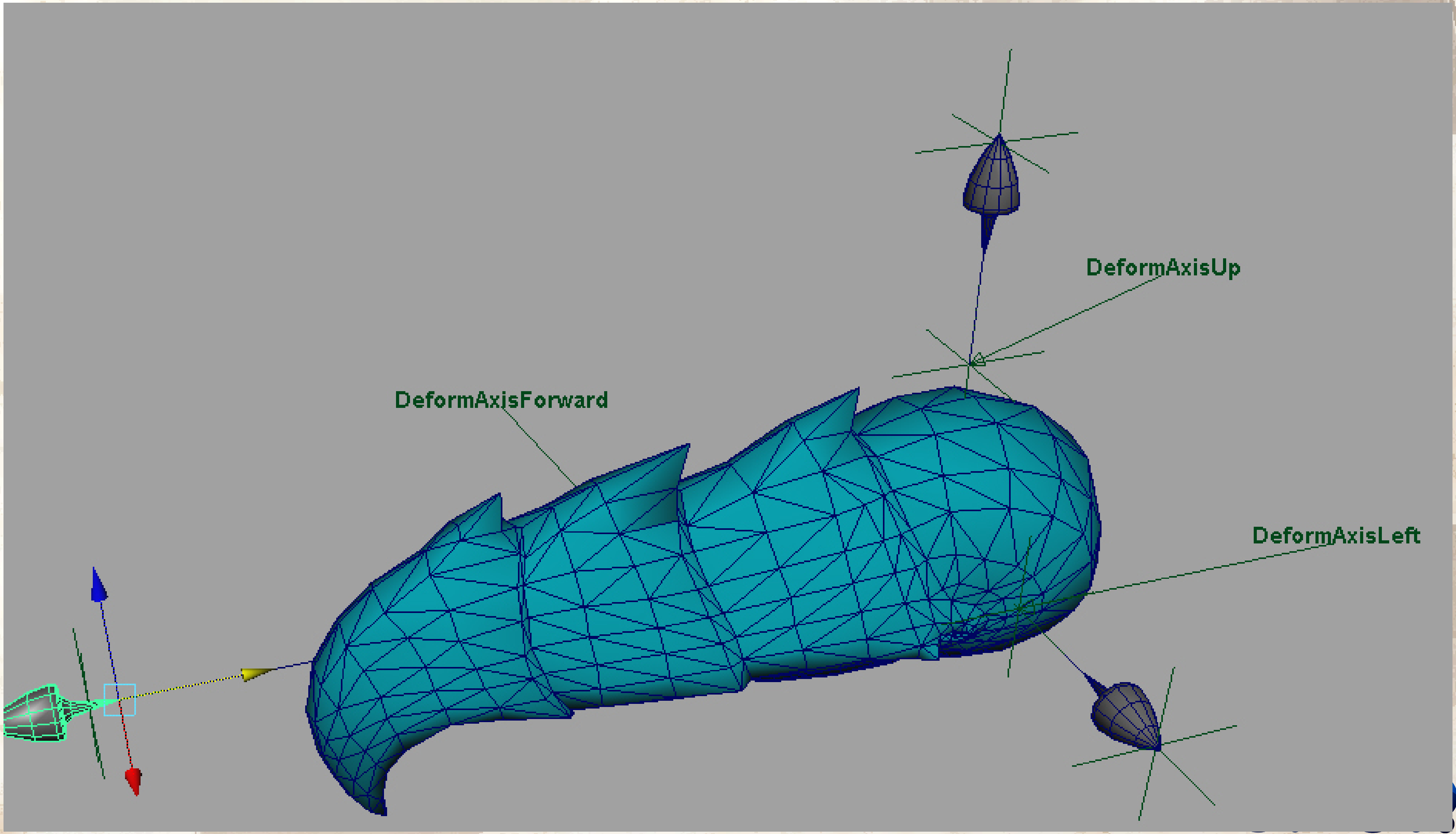
Animation Deforms Mesh



Animation Deforms Mesh



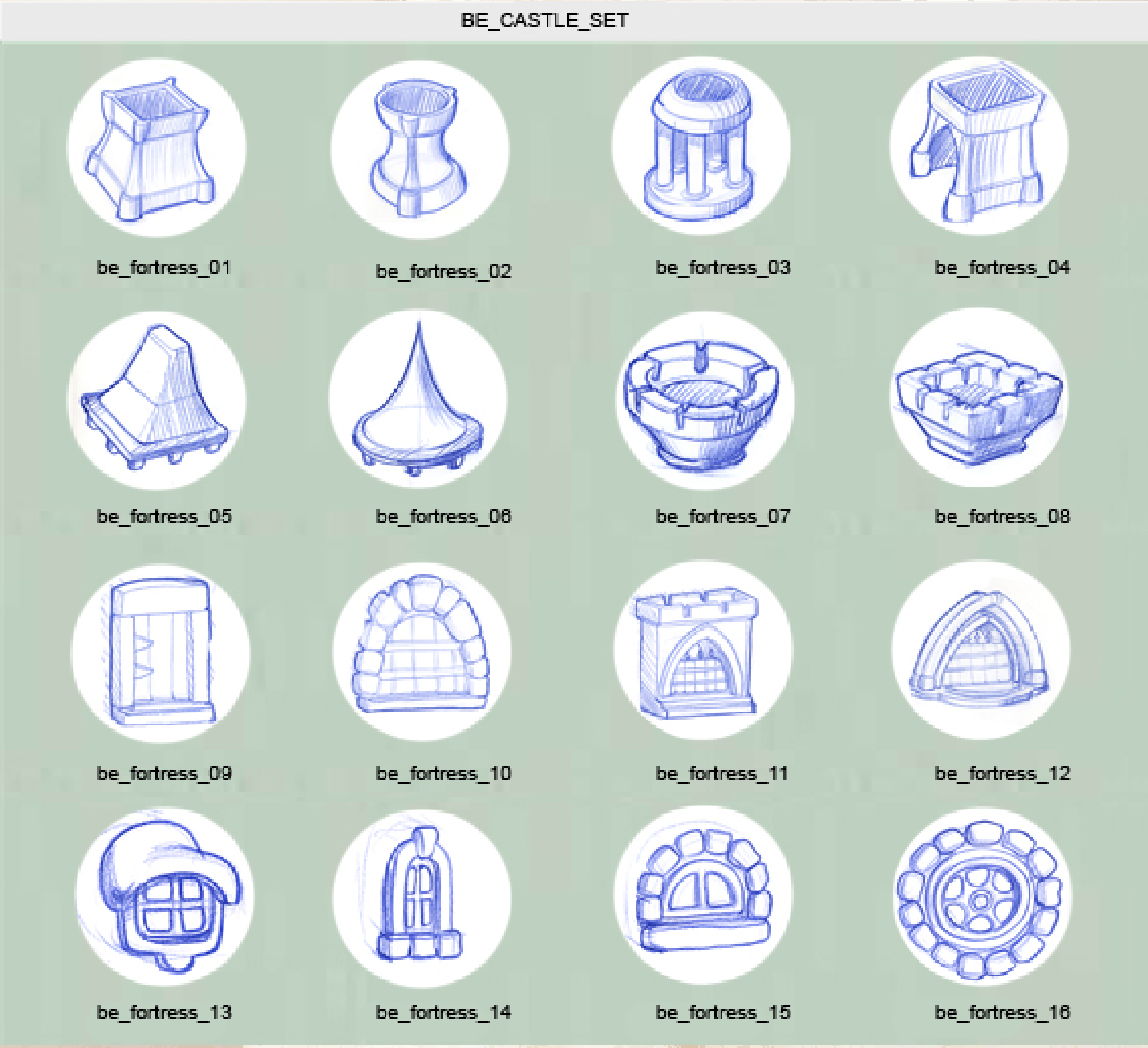
Animation Deforms Mesh



Creature Bodies

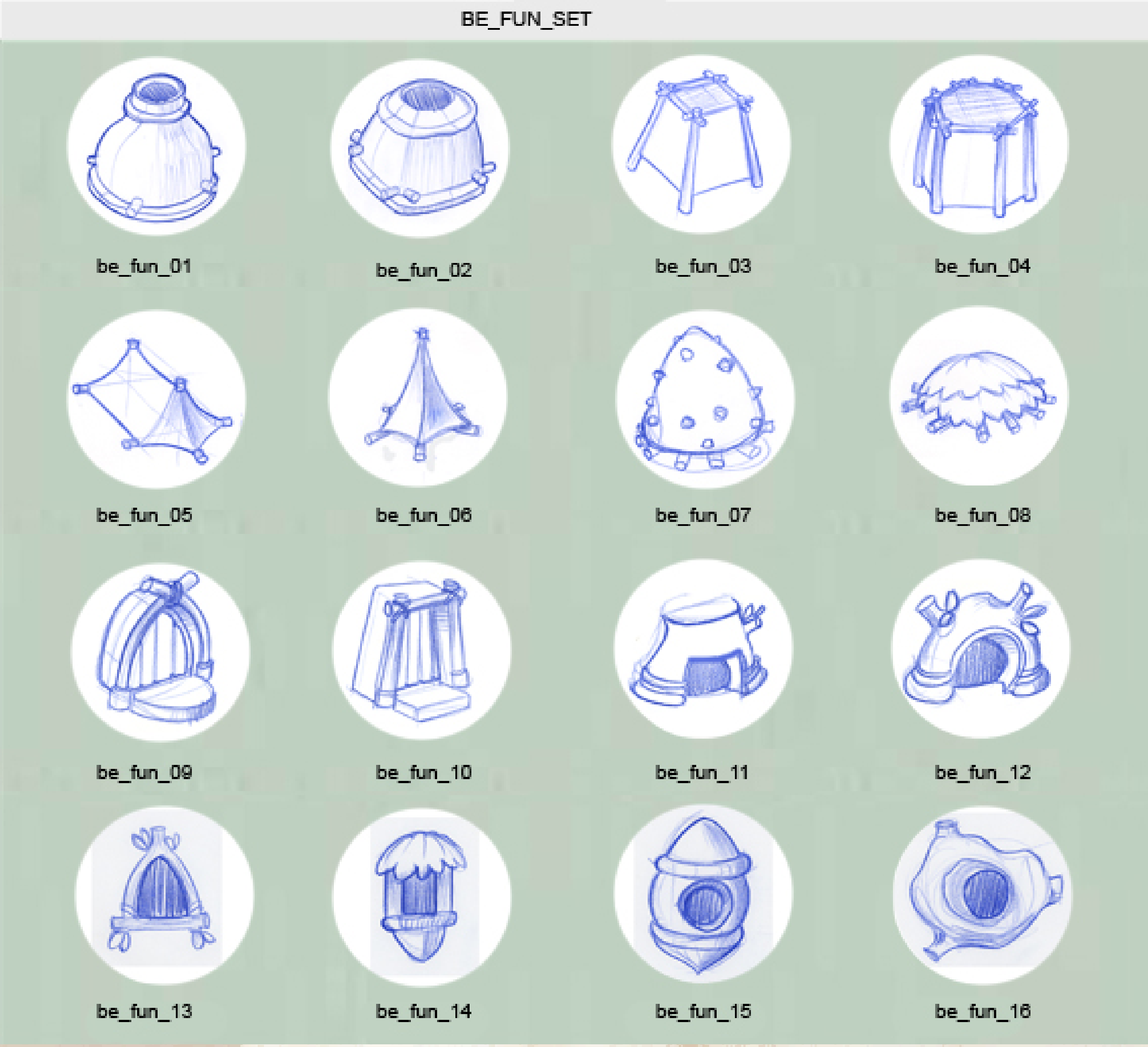
- Base block is a special block: **body mesh**
- Allow player control over a basic skeleton
  - Adjust spline, glue limbs
- Mesh generated via metaballs
- Rigblocks attached to body

Storyboarding

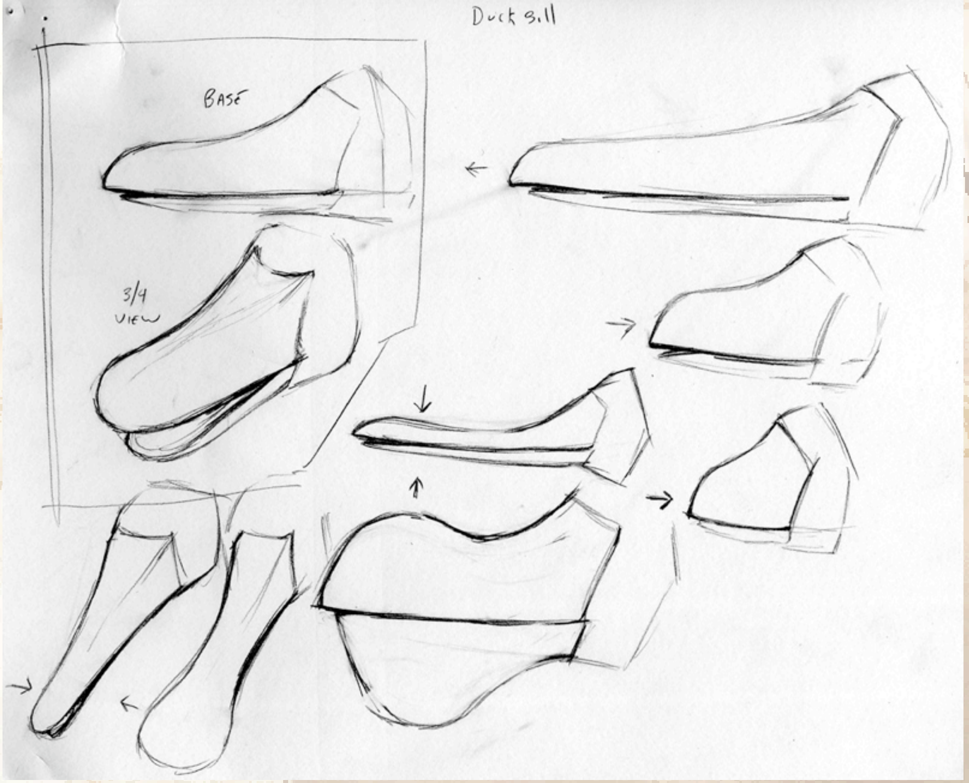


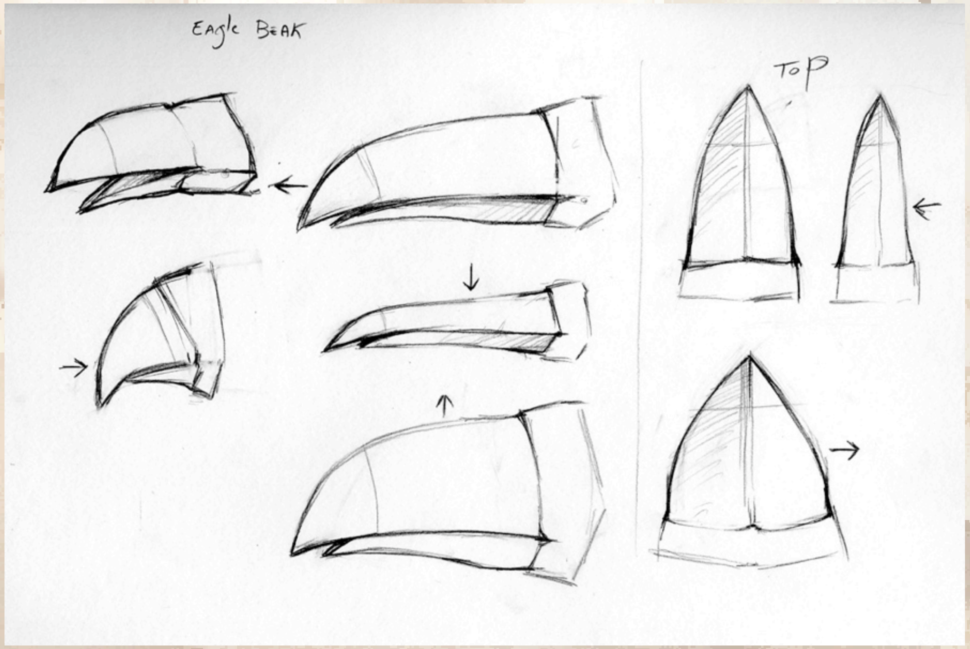
SPORE

Storyboarding



Storyboarding: A Single Block





Pipeline

- Standard workflow: separate author file per animation
- Rigblocks: Multiple animations, so use track editor
- MEL scripts control addition of handle rigs
  - Handles drive animation! (Via expressions)
  - Artist places handle, so can iterate in-Maya

Animation Technology

- Can’t use standard animation blending

50% Def_A + 50% Def_B != Average(A, B)

- Use cumulative blending from rest pose
  - Match Maya by composing deform matrix at end from separately accumulate scale, rotate, translate
- Multiblender
  - Handles standard “runtime” animations
  - Applies deforms on top

Baking

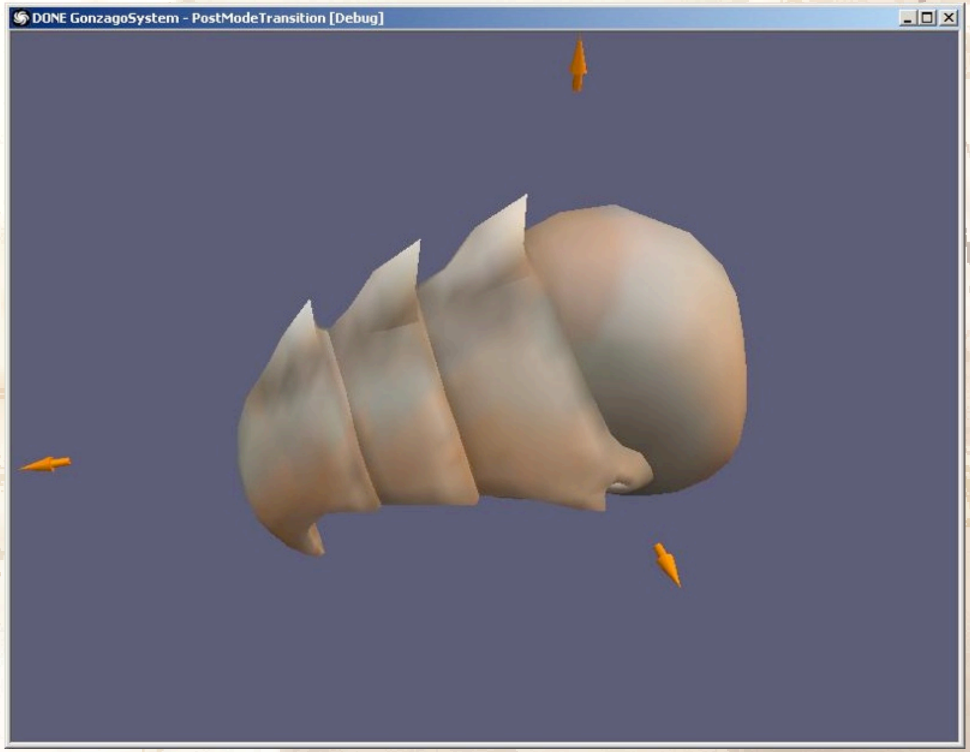
- Remove all deform animations, producing a new base mesh
- Model must be able to be rendered at game rates
  - Single texture page, single material
  - Generate LODs

Baking: Animation

- Desirable for blocks to carry “runtime” animations through (e.g. mouths)
- But such rigblocks must be substituted with low-bone-count versions
- Requires retargetting composite deform pose to new runtime skeleton (base pose has changed)

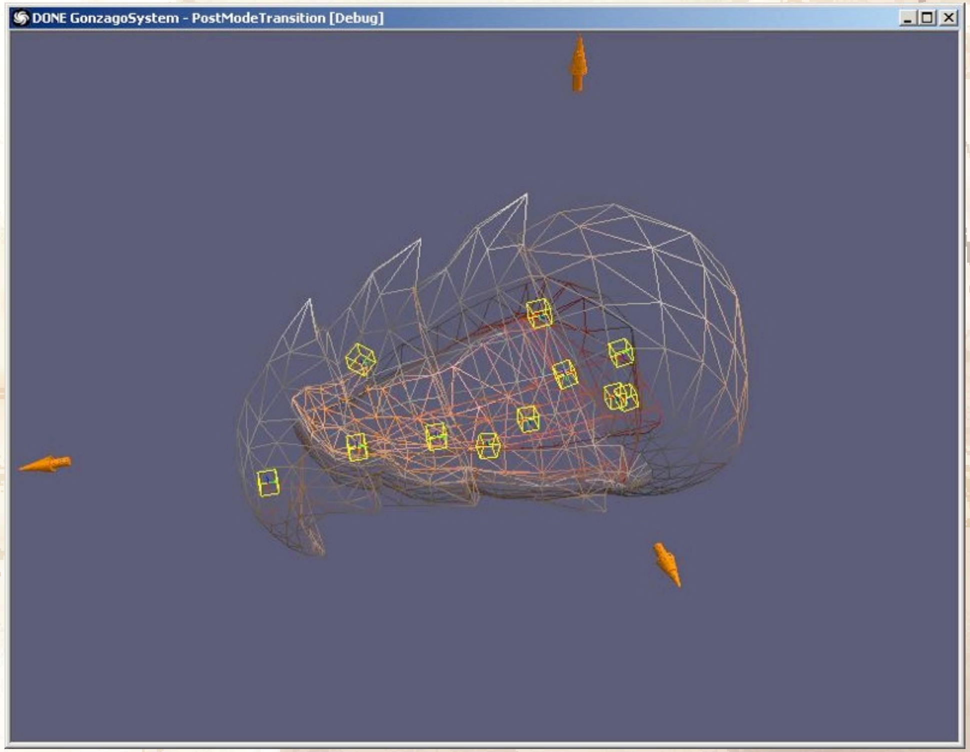
Runtime Animation

Authored Block



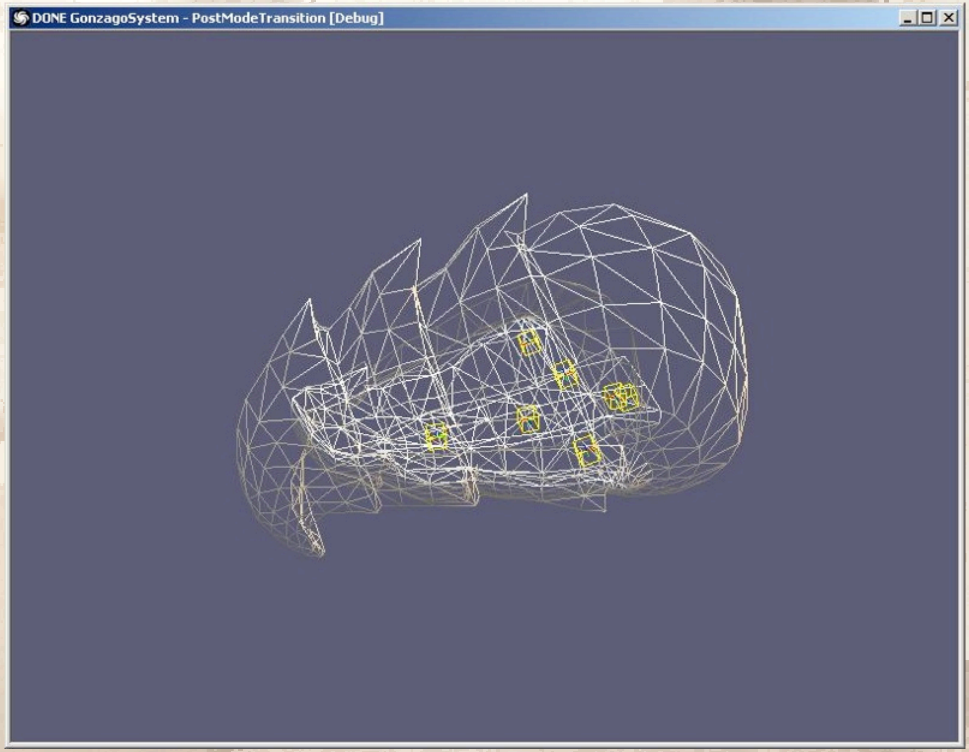
Runtime Animation

Authored Block



- Many bones
- Skeletal animation
- Blendshape animation

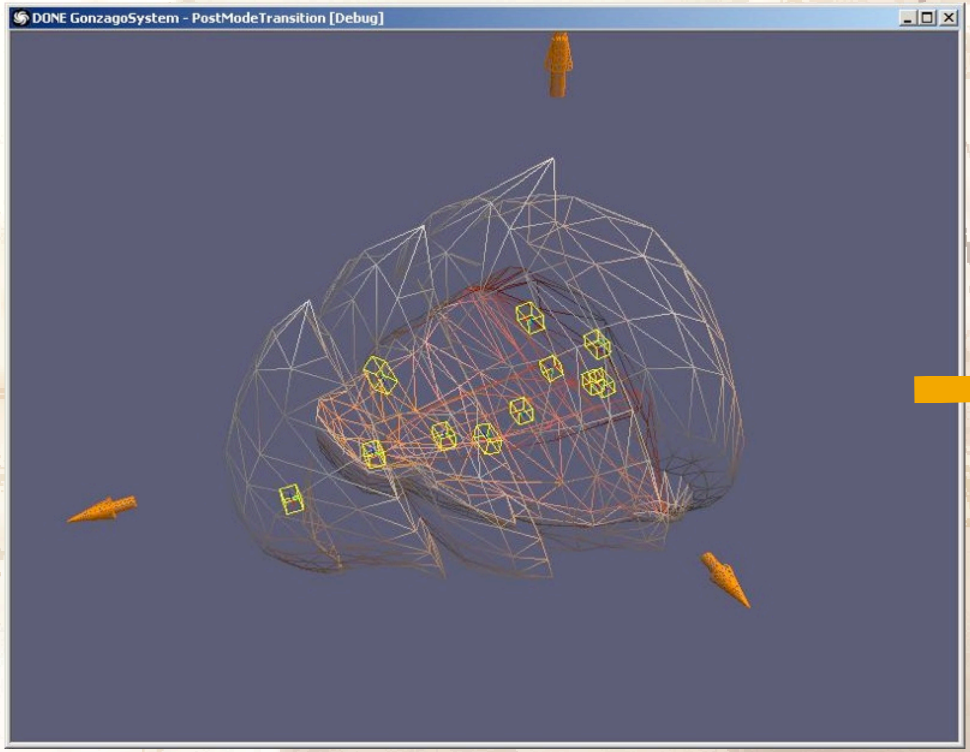
Runtime Block



- Reduced skeleton
- Skeletal animation

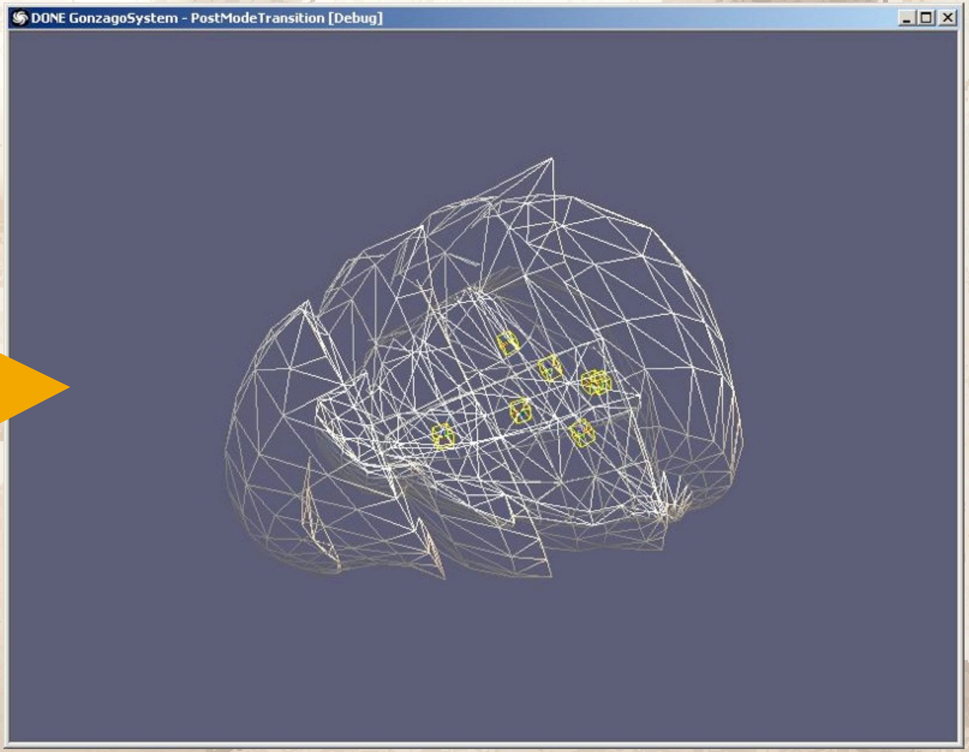
Runtime Animation

Authored Block



- Apply deformation handle

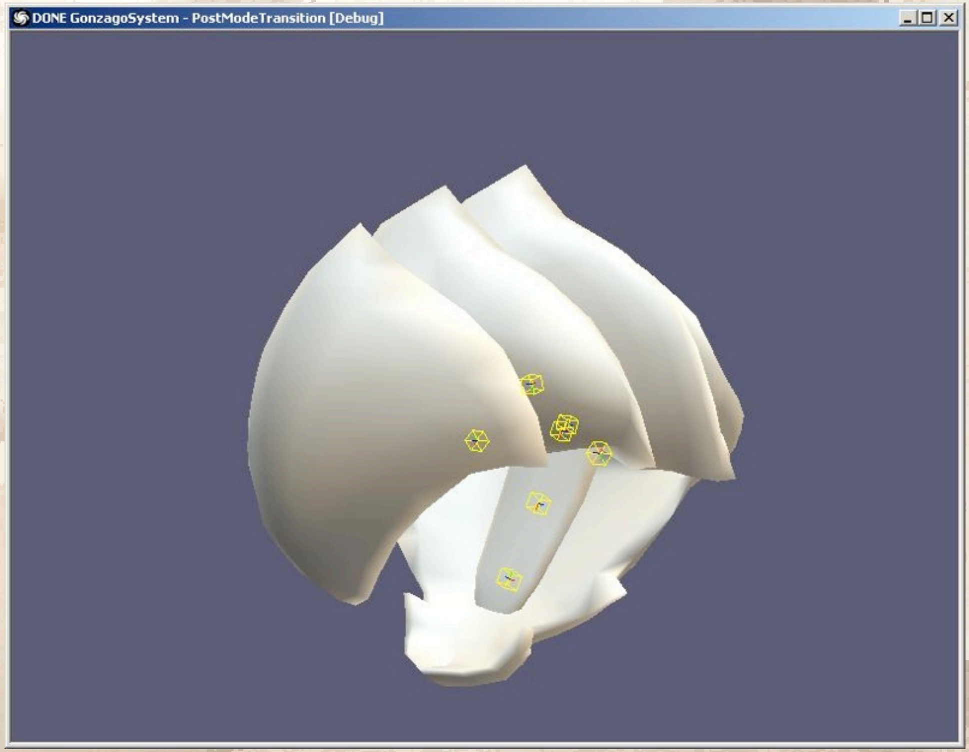
Runtime Block



- Mesh is retargeted to new (runtime) skeleton

Runtime Animation

Runtime Block



- Runtime animations are retargeted to new skeleton

Procedural Creature Animation

SPORE

Kees van Prooijen, Chris Hecker, Jordan Maynard, Ryan Enslow, Bernd Raabe, John DeWeese

John Cimino, Tony Gialdini, Bob King, Gal Roth

Creature Animation

- Standard art pipeline:
  - Modeller makes the asset, with a given skeleton
  - TD sets up a “rig” for that skeleton
  - Animator drives that rig to create animation clips
- Spore:
  - PLAYER makes the model, arbitrary skeleton
  - ???
  - Animator has to animate ... what?
- Profit!

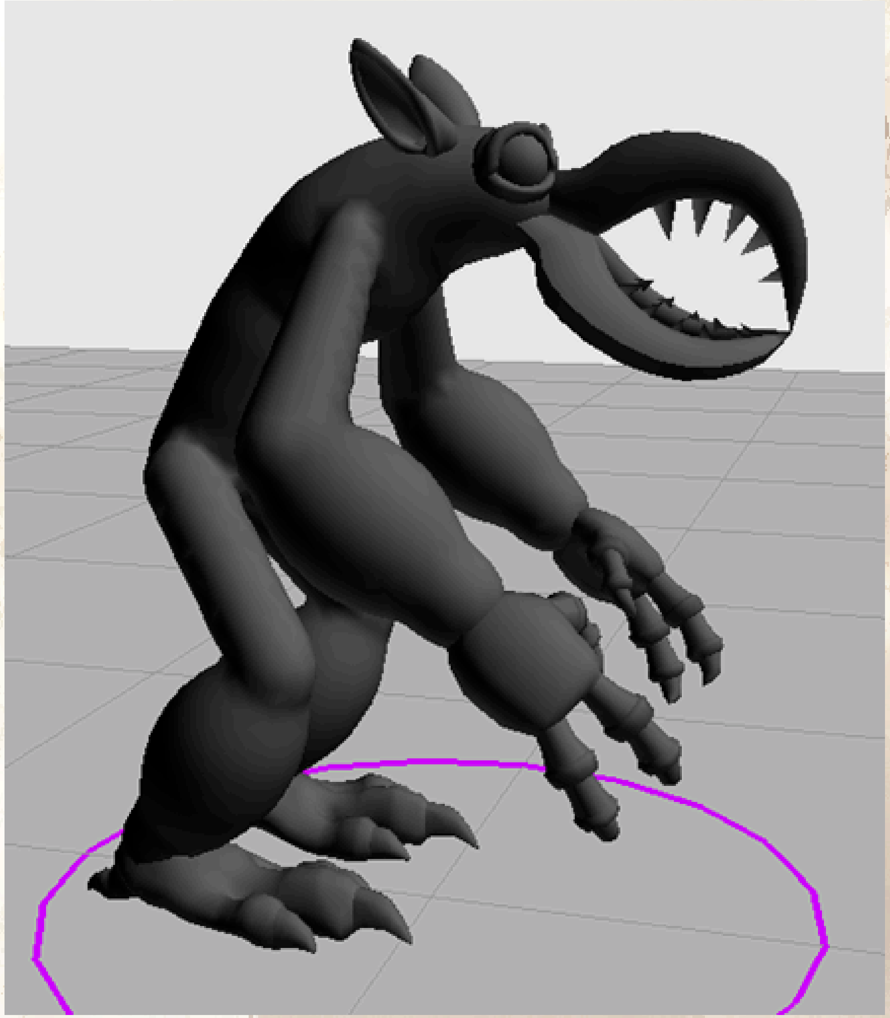
SPORE

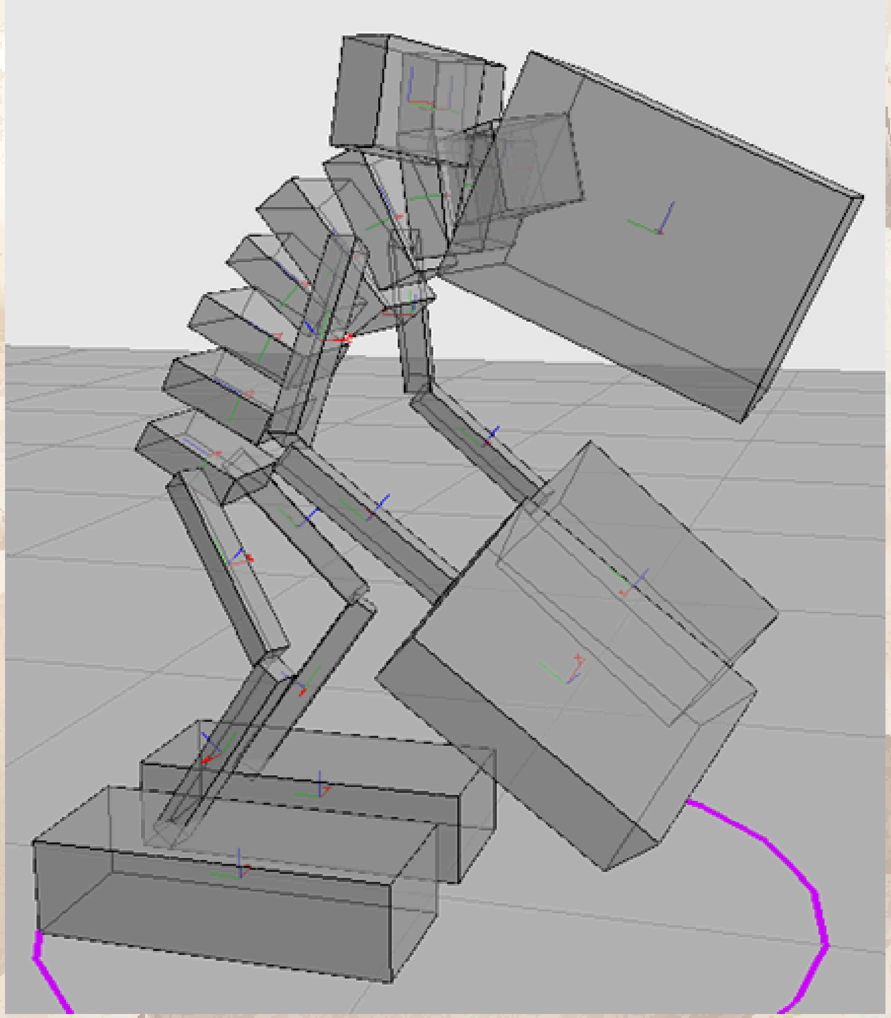
Starting Point

- Creatures are: a spine, with attached limbs, and rigblocks (parts)

- Tree structure
- Know what endpoints are
- Know limbs vs. backbone
- Can attach properties to rigblocks
  - mouth, foot, weapon

Underneath the Skin





History

- Originally:
  - Identified feet/graspers
  - Used (programmer-written) Lua rules to drive them

- Mixed Results
  - Feet worked pretty well
  - Poor quality
  - No way to get an animator in the loop

Havok

- Want to *abstract* animation
  - Just act on those things we know about
  - "Fill in" other bones somehow
- So, run rigid-body physics on skeleton
  - Gives roughly plausible motion
- Add contraints for end effectors
- Good initial results
  - Just driving feet gave natural motion, nice secondary effects

Feet

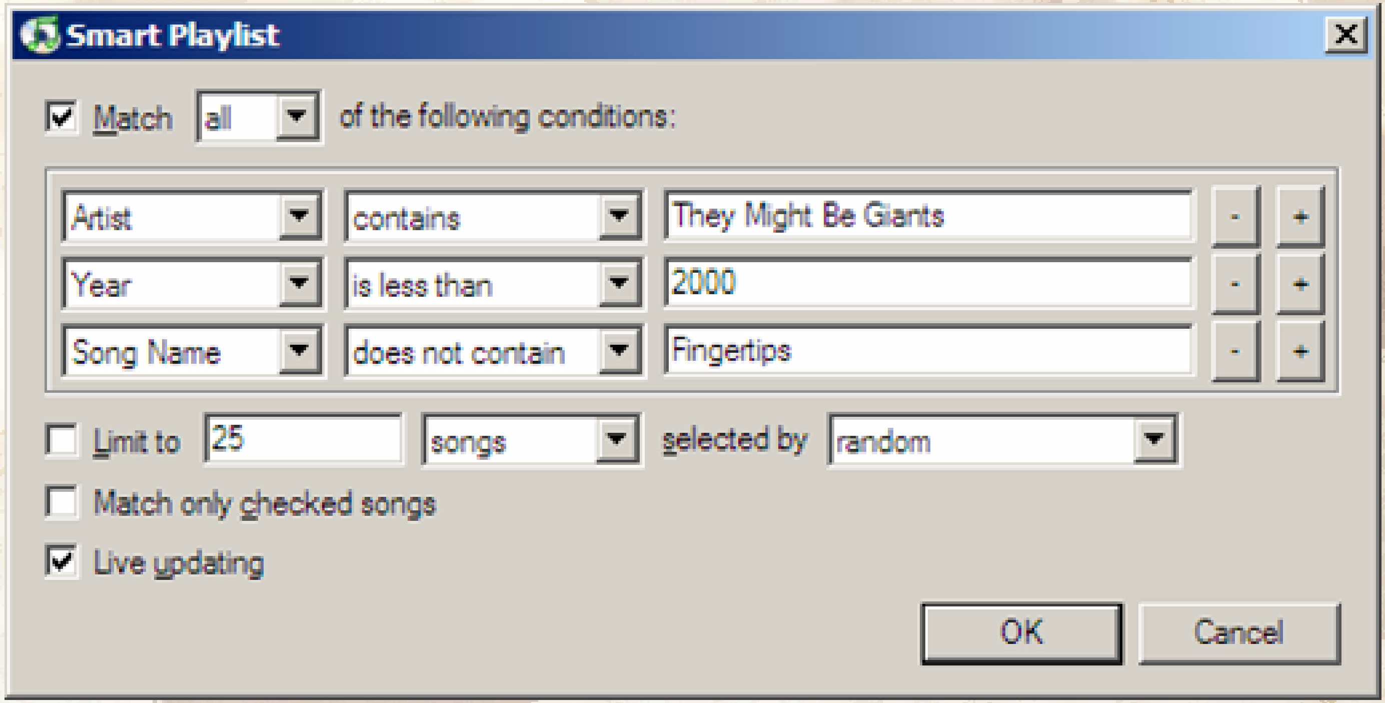
- Started off as simple phase function in time and *N* feet
- By tweaking the function, can get different looks: walk, run, sidle
- Current version also drives foot “look”
  - Toe curl (rigblock animation), ankle joint
- Many, many other improvements -- four years of development

Animator Friendliness

- The more familiar we can make the process of animation creation, the better the results
- Good animators are an invaluable resource
- Animators used to keyframing skeletons
- Need to abstract again:
  - Replace specific bones with **selection** queries
  - Animation is not absolute: relative to a **frame**
- Rest is quite familiar

Selection

- Small, simple matching grammar
- Allows animator to select parts to operate on
- Need to handle zero or N matches



Smart Playlist

☑ Match all of the following conditions:

Artist	contains	They Might Be Giants
Year	is less than	2000
Song Name	does not contain	Fingertips

□ Limit to 25 songs selected by random

□ Match only checked songs

☑ Live updating

Frames

- Even with selection, need some context

- Move <grasper> to same creature’s <mouth>
- Move <grasper> to other creature’s <grasper>
- Move <mouth> to <target>
- Move <weapon> to ground-relative <target>

- Animations keyed with respect to frame

Havok?

- Worked well for exploration
  - helped us clarify what we needed the system to do
- However, had **poseability** problems
  - Animator could set up a pose that the creature had problems reaching
  - The more complicated (and thus constrained) the animation, the worse the behaviour
  - Needed to re-think approach or face uncomfortable limits on artist creativity

Havok Replacement

- Forget physics + constraints, too sloppy: use direct IK
- Wrote our own specialized IK solver
- Several iterations to get right
- Final one that worked: particle-based IK
- Plus, lots of semantic info, special cases, tweaking
- Simplicity of particle approach lends itself to this

Secondary Motion

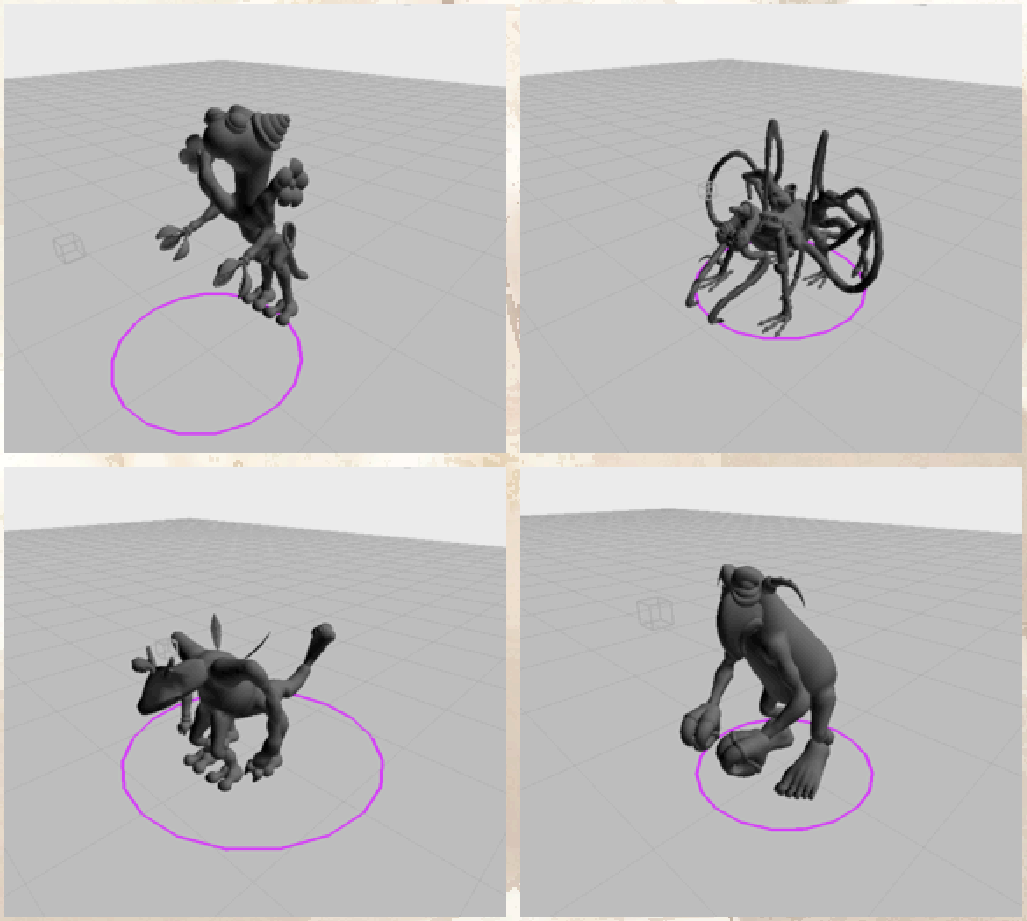
- Problem: lost all our secondary motion! No more swaying, natural follow-along movement
- Layered new system on top to re-inject this: Wiggles
- Finds bodies that aren’t driven directly, and uses spring mass system to add secondary movement

Part Animation

- 50-90% of creature is *parts*, not interior skeleton
- Rigblocks have runtime animations
- Procedural animation system can trigger (or scrub) these
  - Open/close mouth animation
  - Graspers open/close/point
- Also: trigger effects on bones

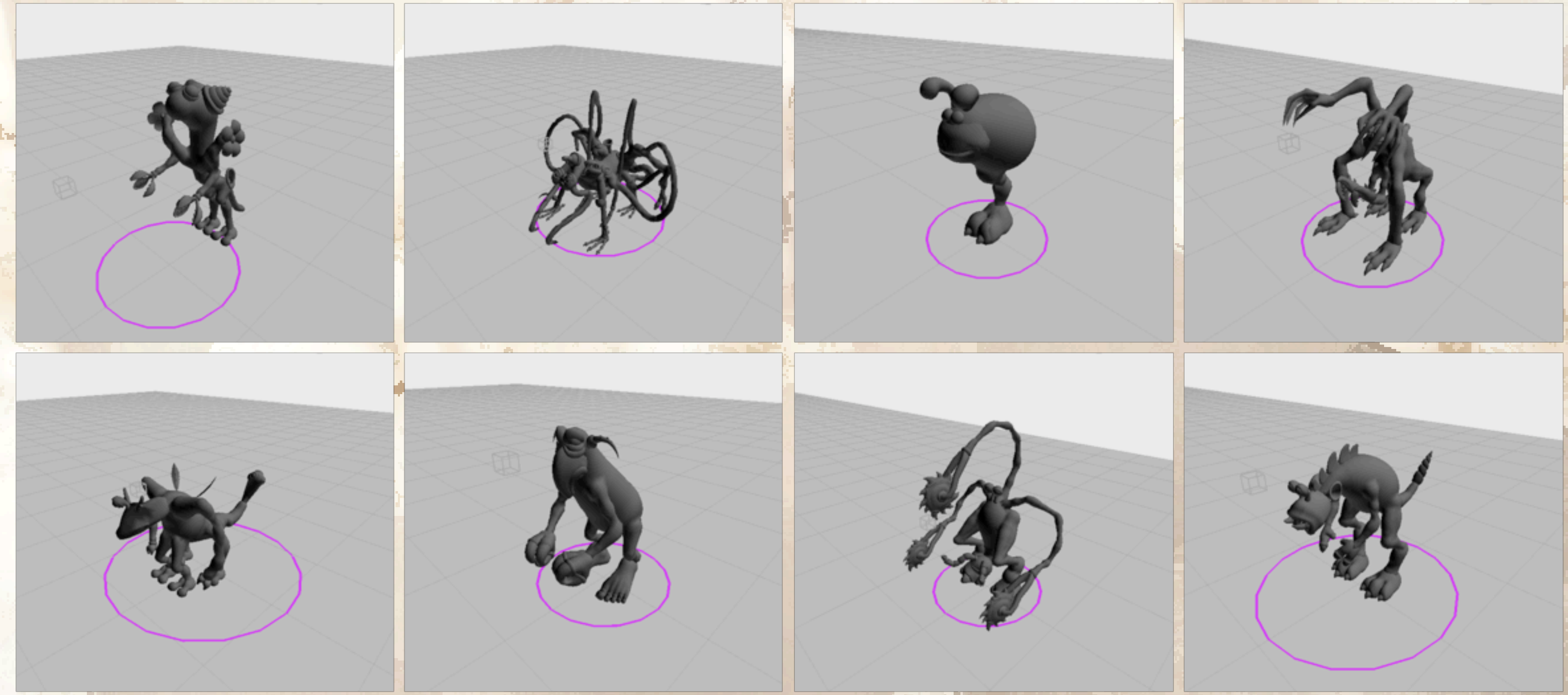
Remaining Problems

- Iteration, Iteration, Iteration....

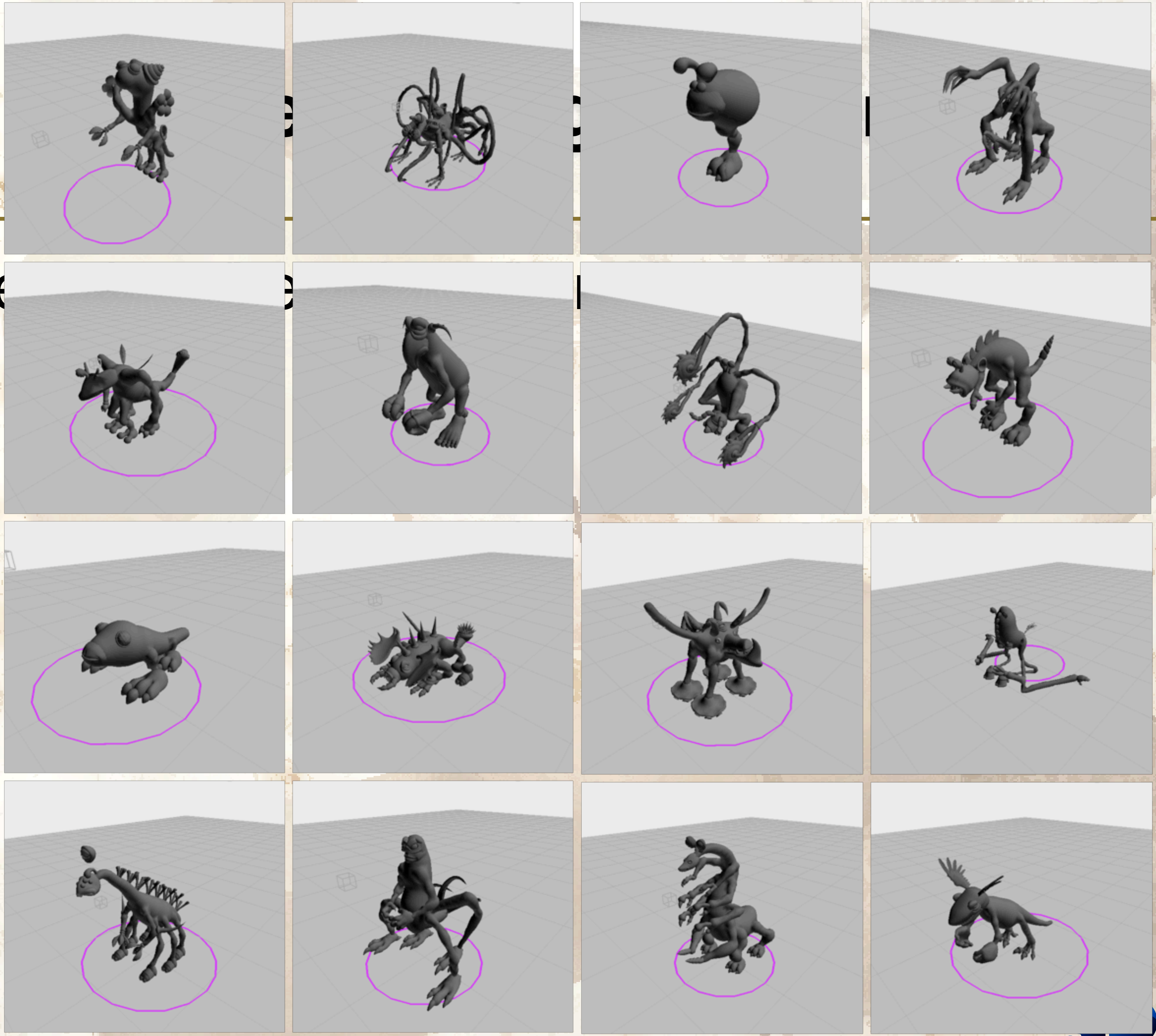


Remaining Problems

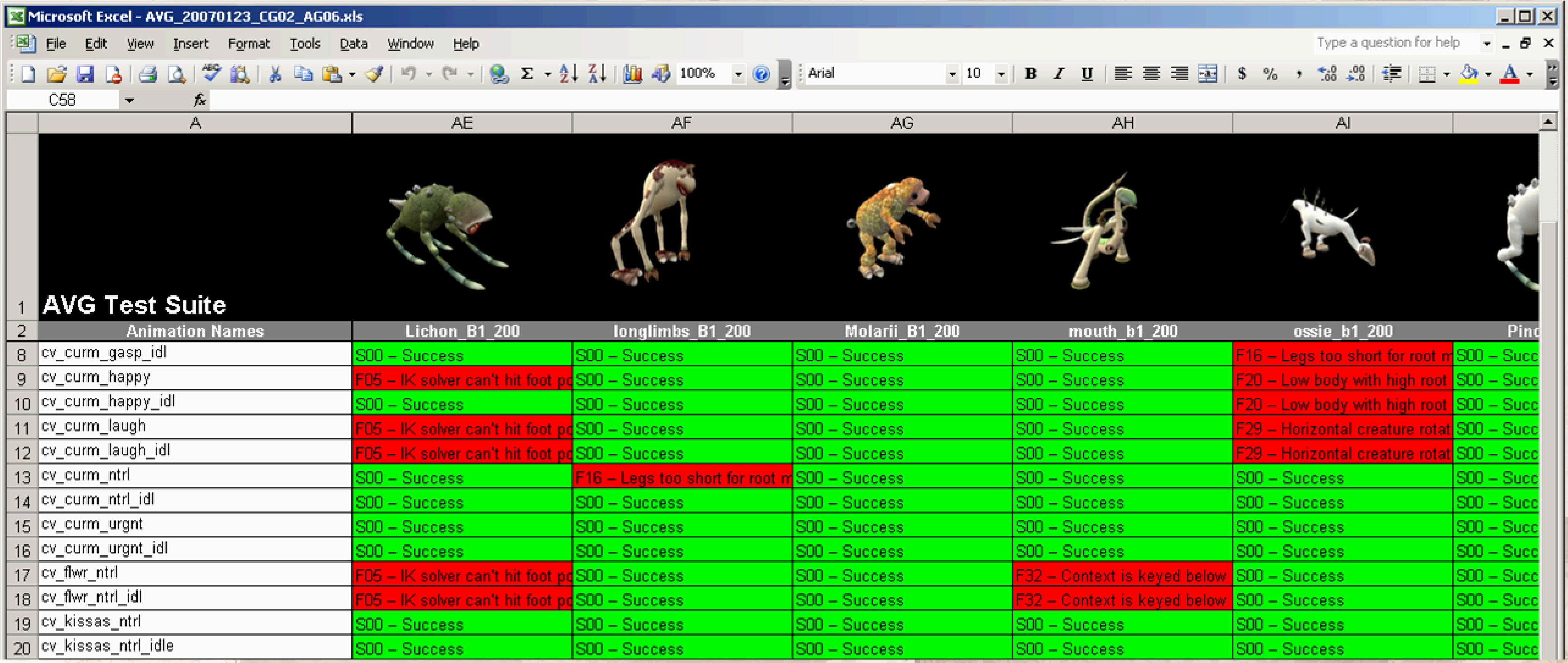
- Iteration, Iteration, Iteration....



- Ite



AVG



Microsoft Excel - AVG_20070123_CG02_AG06.xls

	A	AE	AF	AG	AH	AI	
1	AVG Test Suite						
2	Animation Names	Lichon_B1_200	longlimbs_B1_200	Molarii_B1_200	mouth_b1_200	ossie_b1_200	Pinc
8	cv_curm_gasp_idl	S00 – Success	S00 – Success	S00 – Success	S00 – Success	F16 – Legs too short for root m	S00 – Succ
9	cv_curm_happy	F05 – IK solver can't hit foot pc	S00 – Success	S00 – Success	S00 – Success	F20 – Low body with high root	S00 – Succ
10	cv_curm_happy_idl	S00 – Success	S00 – Success	S00 – Success	S00 – Success	F20 – Low body with high root	S00 – Succ
11	cv_curm_laugh	F05 – IK solver can't hit foot pc	S00 – Success	S00 – Success	S00 – Success	F29 – Horizontal creature rotat	S00 – Succ
12	cv_curm_laugh_idl	F05 – IK solver can't hit foot pc	S00 – Success	S00 – Success	S00 – Success	F29 – Horizontal creature rotat	S00 – Succ
13	cv_curm_ntrl	S00 – Success	F16 – Legs too short for root m	S00 – Success	S00 – Success	S00 – Success	S00 – Succ
14	cv_curm_ntrl_idl	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Succ
15	cv_curm_urgnt	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Succ
16	cv_curm_urgnt_idl	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Succ
17	cv_flwr_ntrl	F05 – IK solver can't hit foot pc	S00 – Success	S00 – Success	F32 – Context is keyed below	S00 – Success	S00 – Succ
18	cv_flwr_ntrl_idl	F05 – IK solver can't hit foot pc	S00 – Success	S00 – Success	F32 – Context is keyed below	S00 – Success	S00 – Succ
19	cv_kissas_ntrl	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Succ
20	cv_kissas_ntrl_idle	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Success	S00 – Succ

Iteration

- Tight loop between animators and animation engineers
  - Finding problematic situations
  - Fixing by adding new context queries, adding additional IK features
- Very common in game development
  - Keeping iteration overhead low is crucial
  - Hence custom anim tool: Spasm
  - Fast startup, animators/engineers work in same app

Player-Driven Procedural Texturing

SPORE

Henry Goffin, Grue, Chris Hecker, Ocean Quigley, Shalin Shodhan, Andrew Willmott

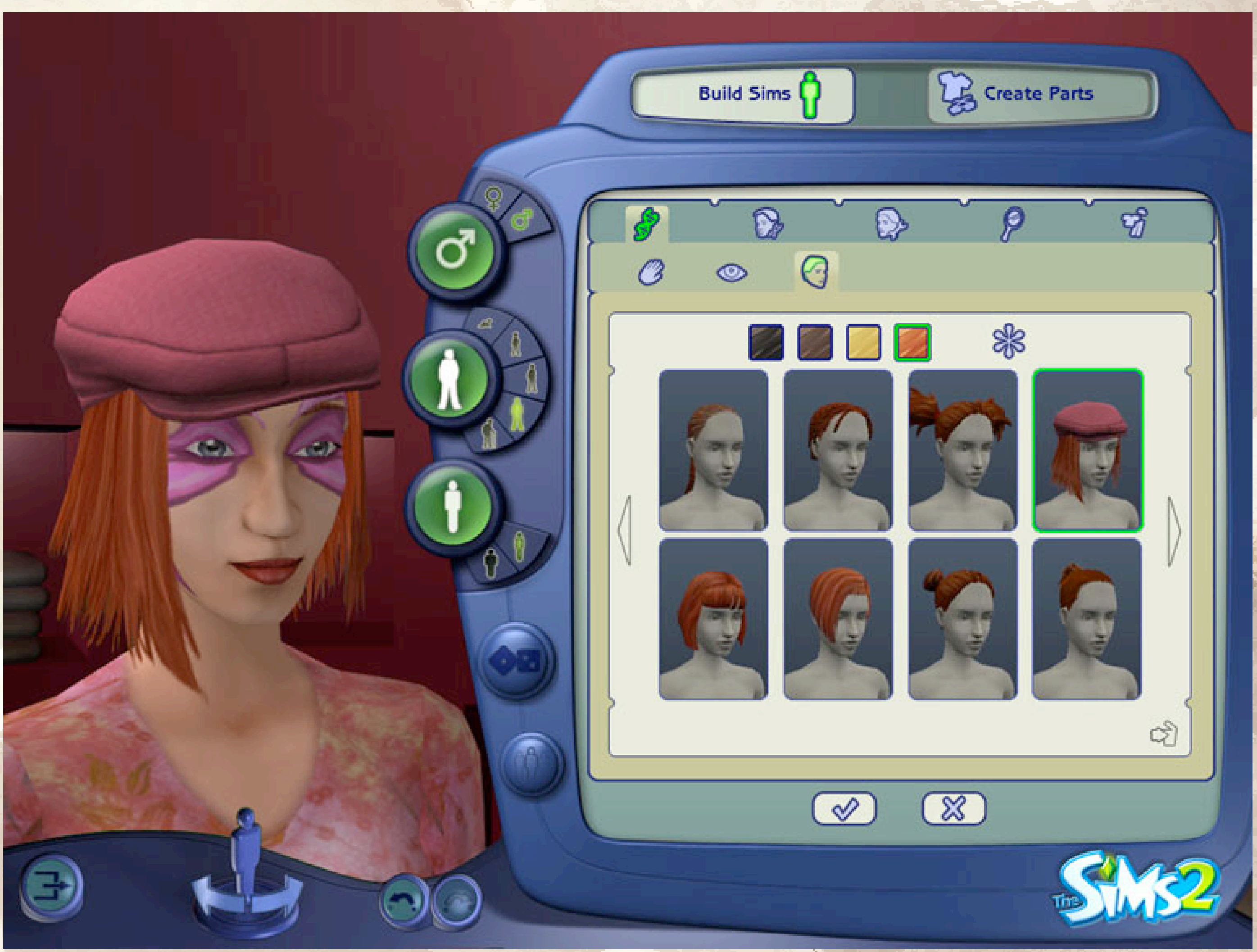
Problem Area

- We let players create their own creatures, huts, buildings, cars, boats, planes...
- How do we texture them?
- Once we’re done, how do we turn this into a game model?

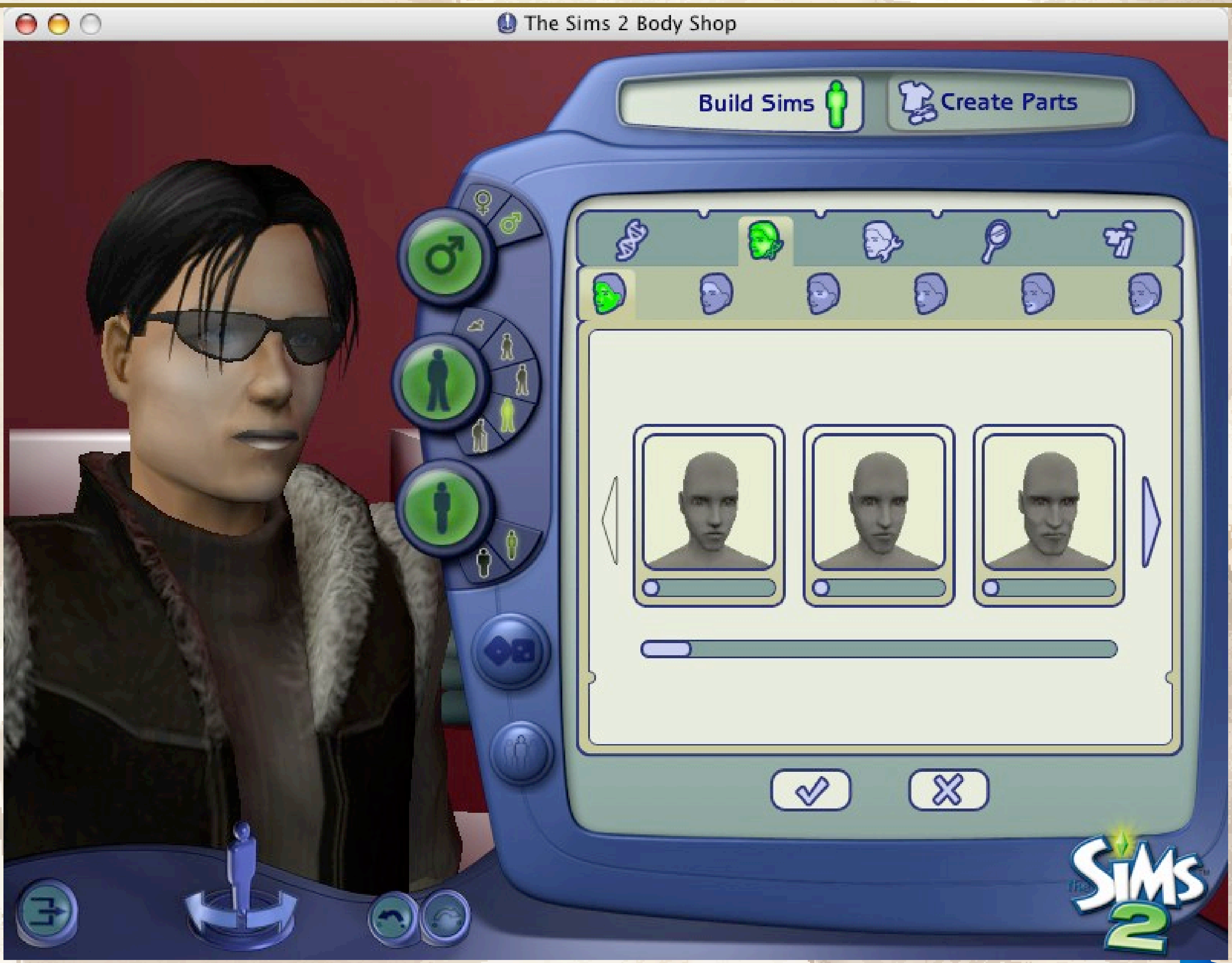
Previous Work

- SSX
  - Swap in different player meshes, accessories
- Sims 2 Bodyshop
  - Facial morphs
  - Select clothing: top and bottom texture pages
- Need for Speed vehicles
  - Decals, morphs on many parts
- Many more

Previous Work



Previous Work



Texturing: Player Control

- Want satisfying player input
- Not too detailed
  - Too tedious for the majority
- Not too simplistic
  - Everyone’s model looks the same
  - Want to go beyond overall texture layer selection

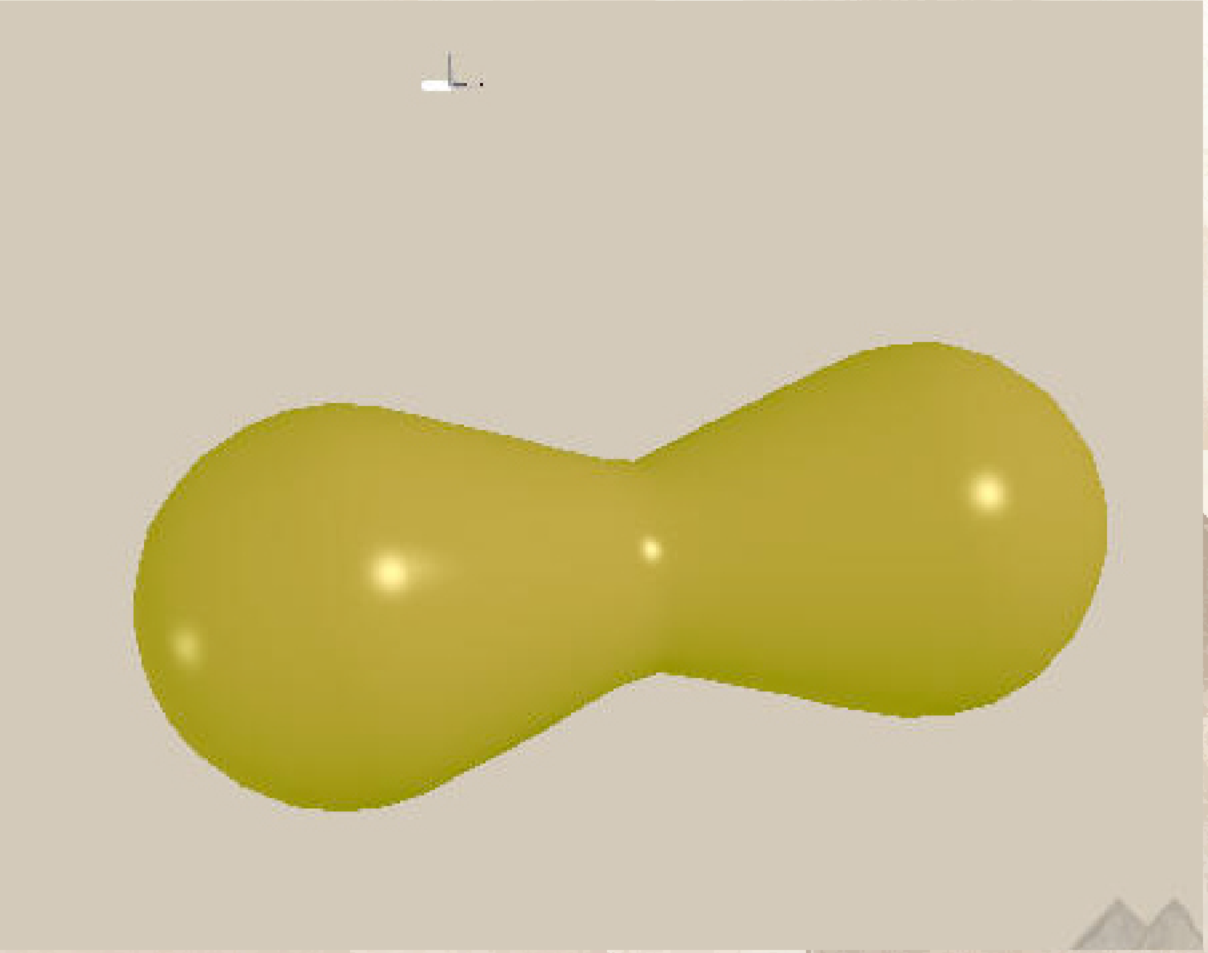
Two Solutions

- 1. Creature Texturing
  - Full brush-driven 3D texture painting
  - Driven by our effects system rather than a human
  - Variety by layering different parameterised scripts
  - Organic look
- 2. “Mineral” Texturing
  - Repeating textures
  - Paint regions
  - Procedural UV’ing
  - Buildings, Vehicles, UFOs

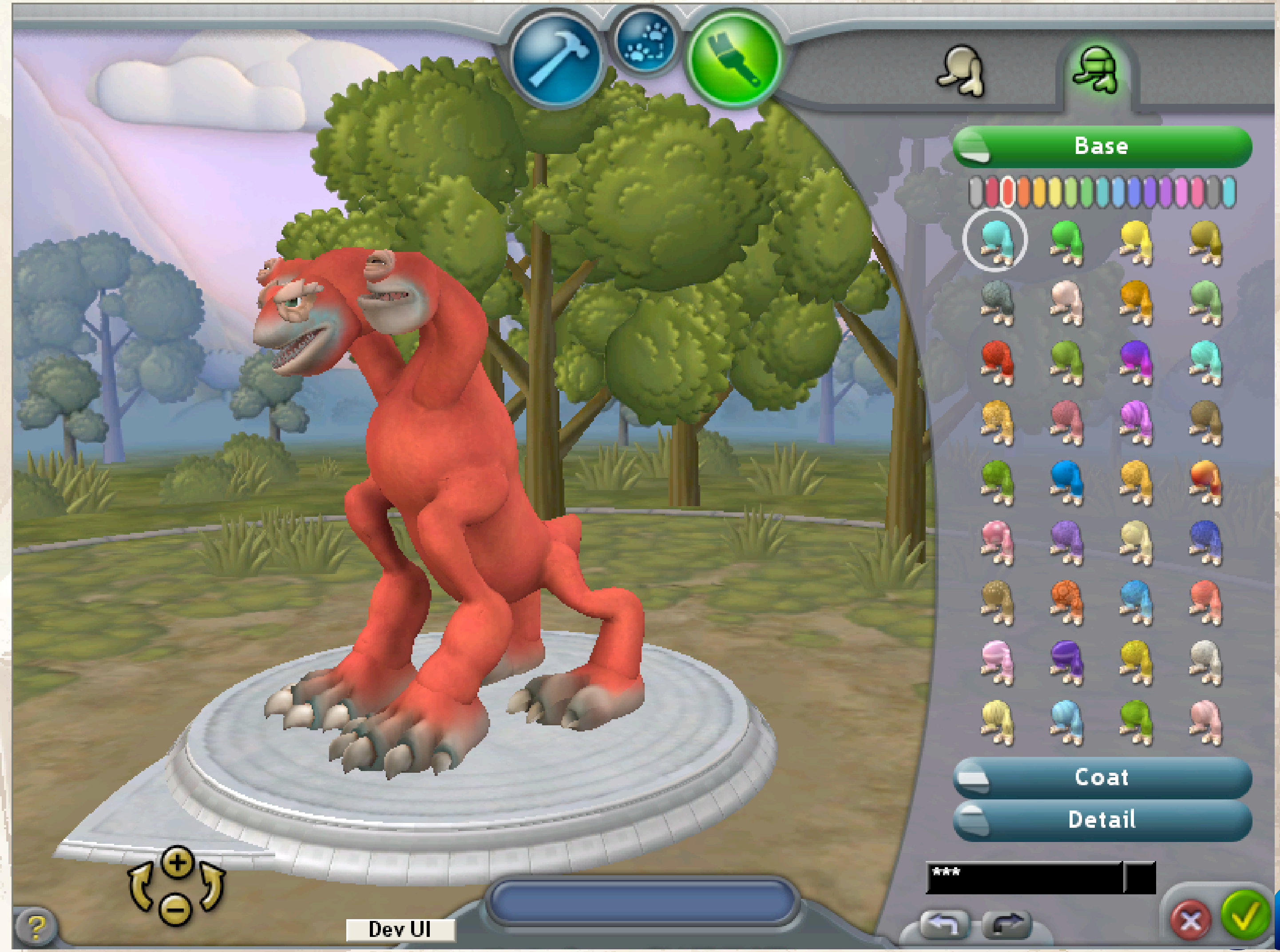
1: Skinpaint

- Brushes: diffuse, spec, alpha, bump map
  - Mesh is uv mapped, for any point on mesh, brush can be splatted into destination texture
- Brush selection and position controlled by effect system
  - "Particles" can be moved over surface using frame adjustment, affected by skeleton
  - Library of effect scripts

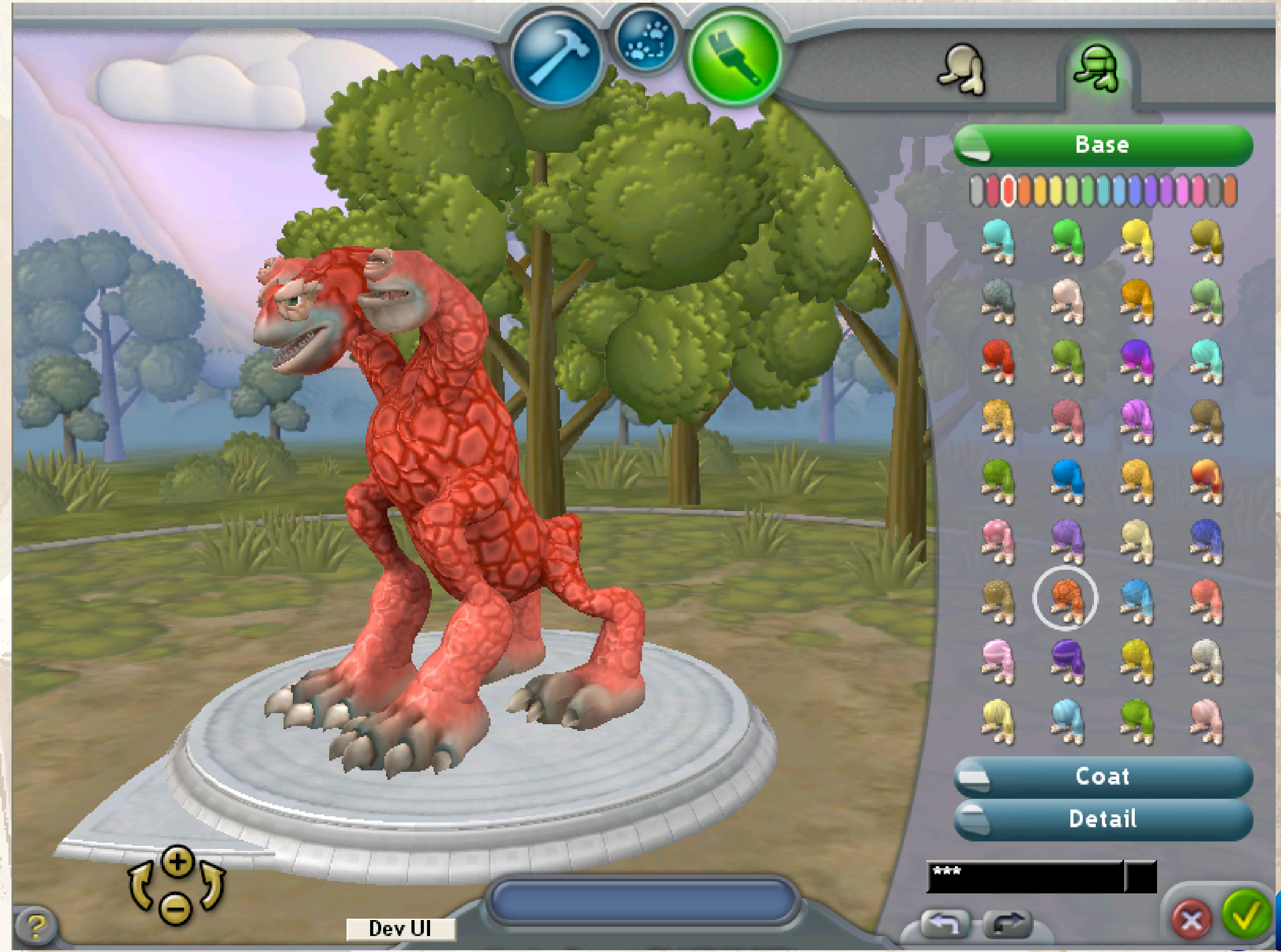
Skinpaint: Early Prototype



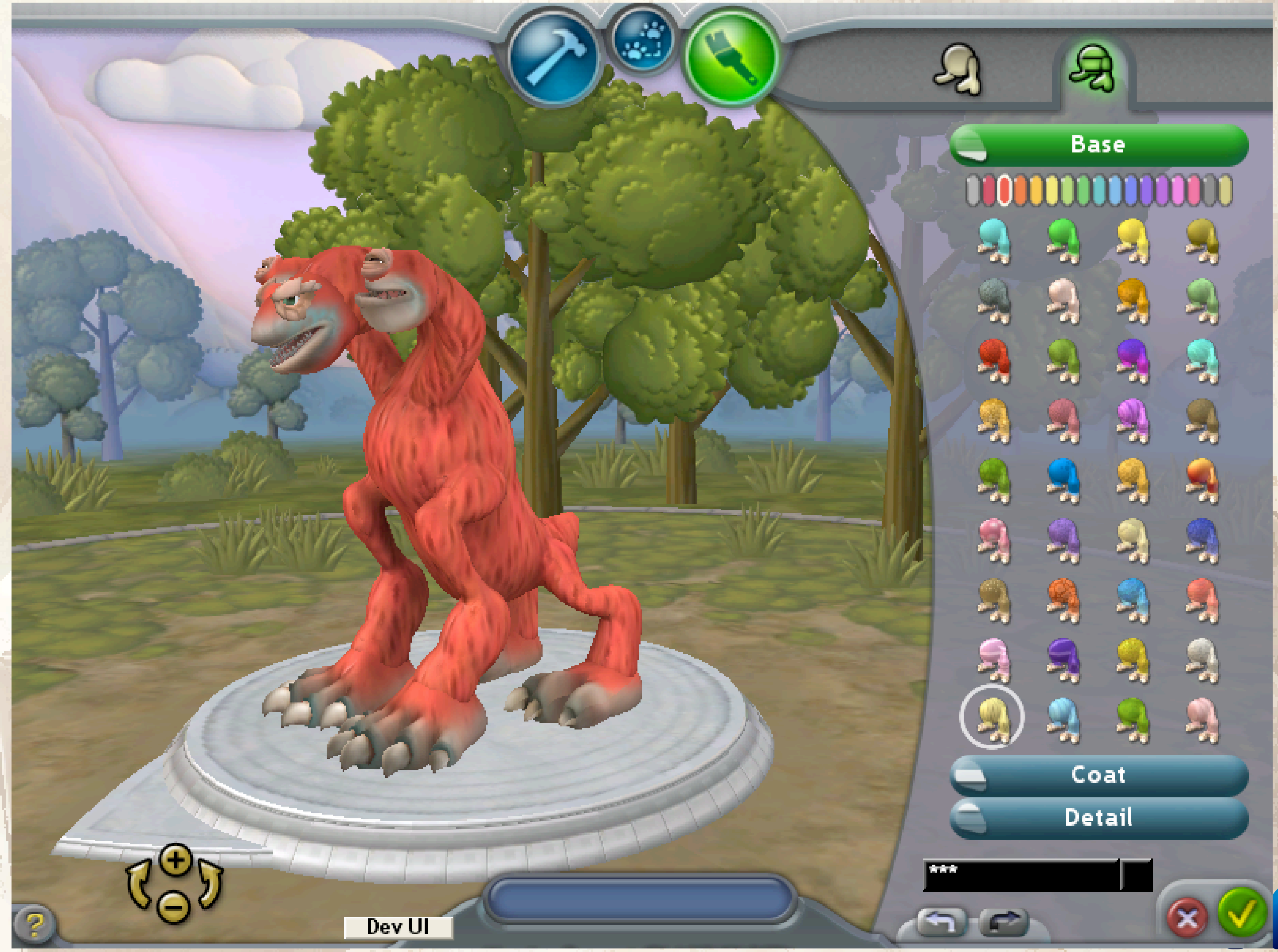
Skinpaint



Skinpaint

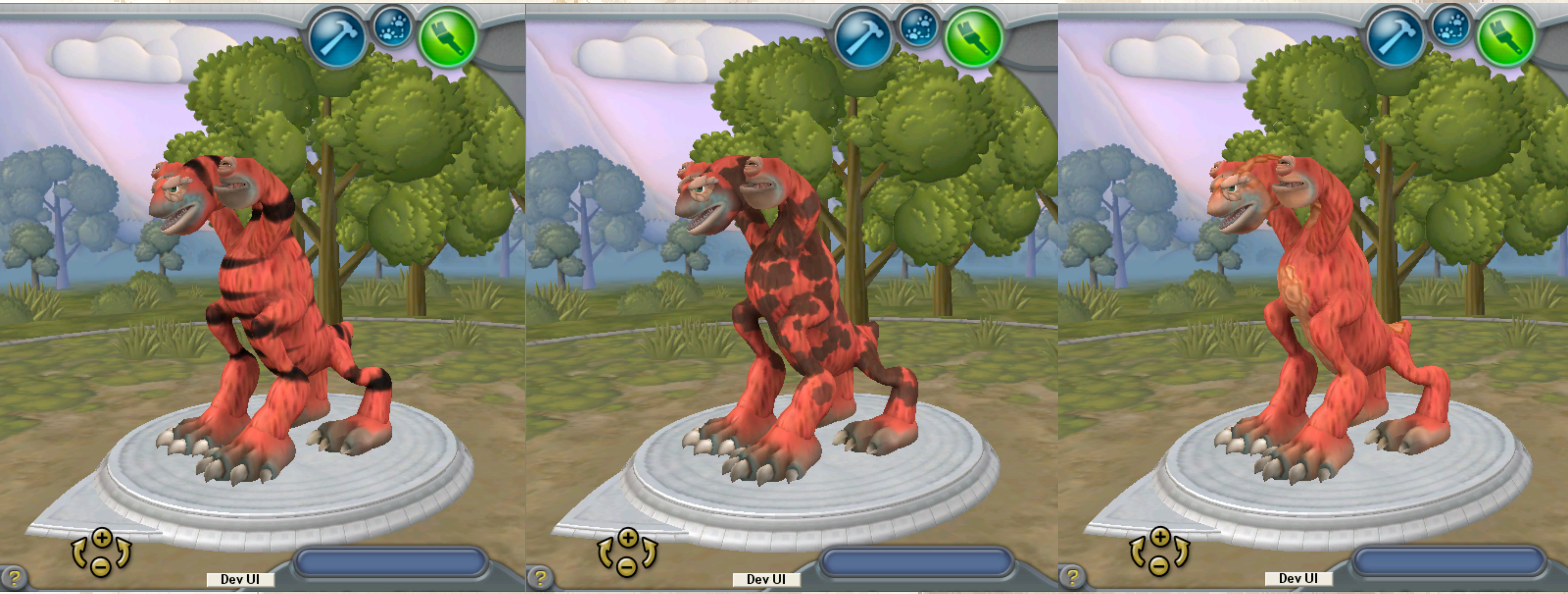


Skinpaint

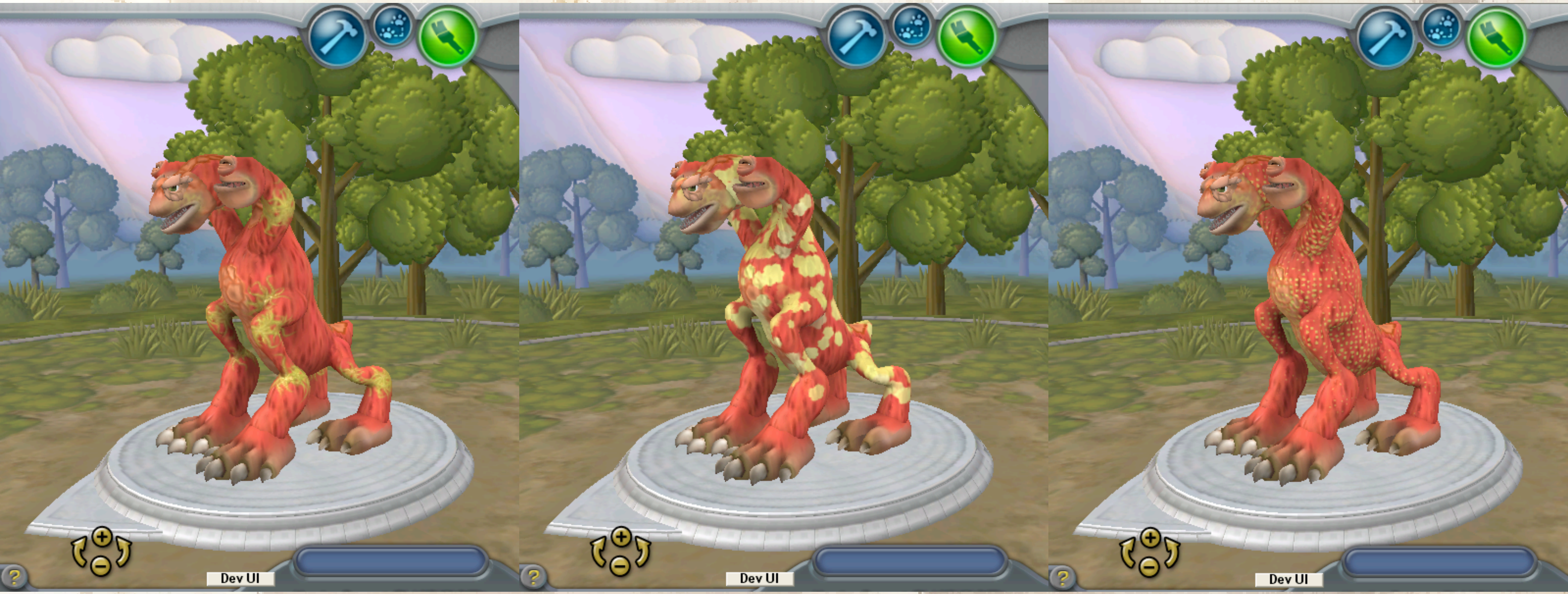


SPORE

Skinpaint: +Coat



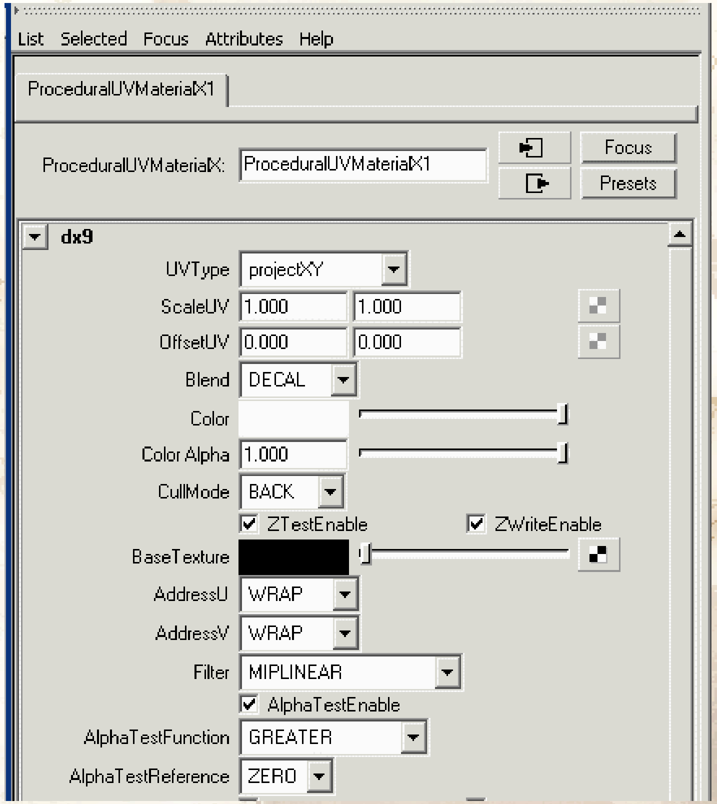
Skinpaint: +Detail



2: Procedural UV’ing

- Model parts deform (Rigblocks)
- Model parts tagged with regions
  - Use for material and functional areas too
- Regions tagged with uv’ing type
  - Boxmap
  - Cylinder, sphere, planar, disc
- Applied in Vertex Shader
- Textures parameterized by two colours

Procedural UV'ing



ProceduralUVMaterialX: ProceduralUVMaterialX1

UVType: projectXY

ScaleUV: 1.000 1.000

OffsetUV: 0.000 0.000

Blend: DECAL

Color

Color Alpha: 1.000

CullMode: BACK

☑ ZTestEnable ☑ ZWriteEnable

BaseTexture

AddressU: WRAP

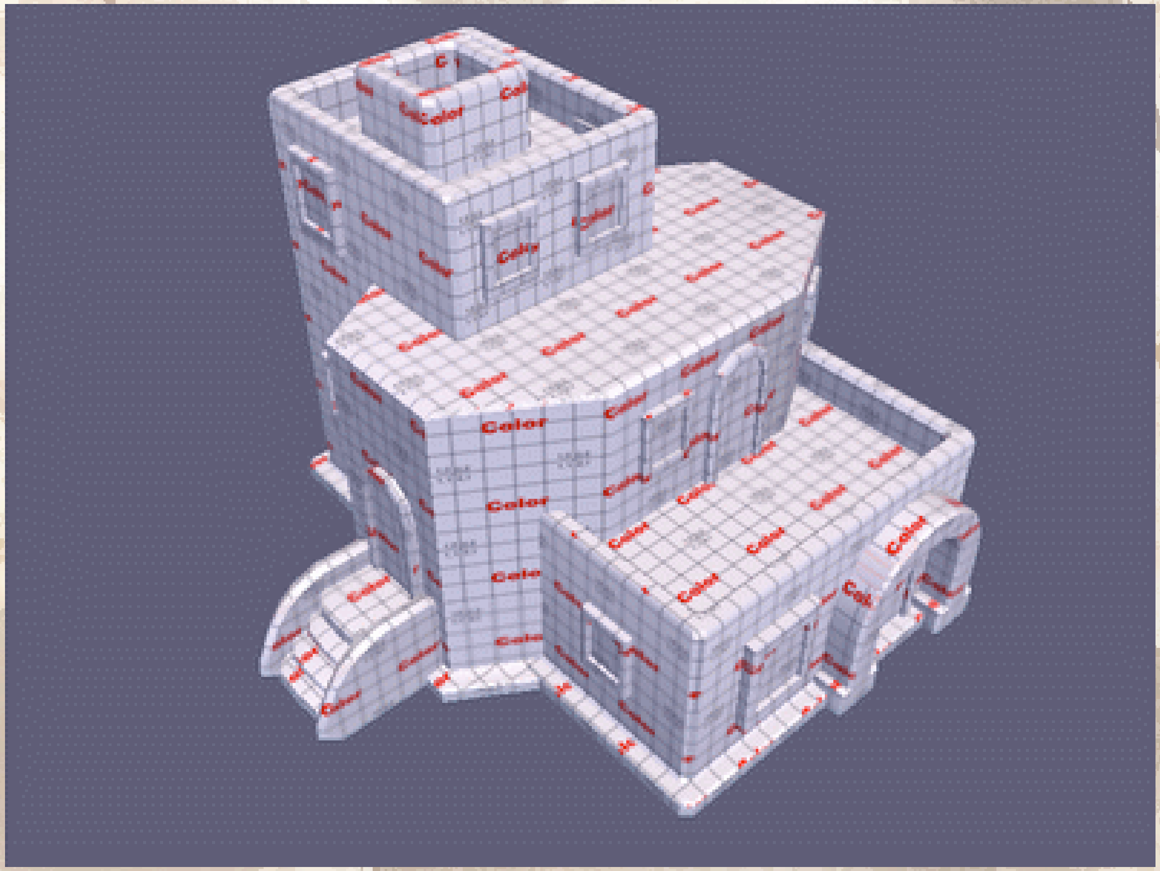
AddressV: WRAP

Filter: MIPLINEAR

☑ AlphaTestEnable

AlphaTestFunction: GREATER

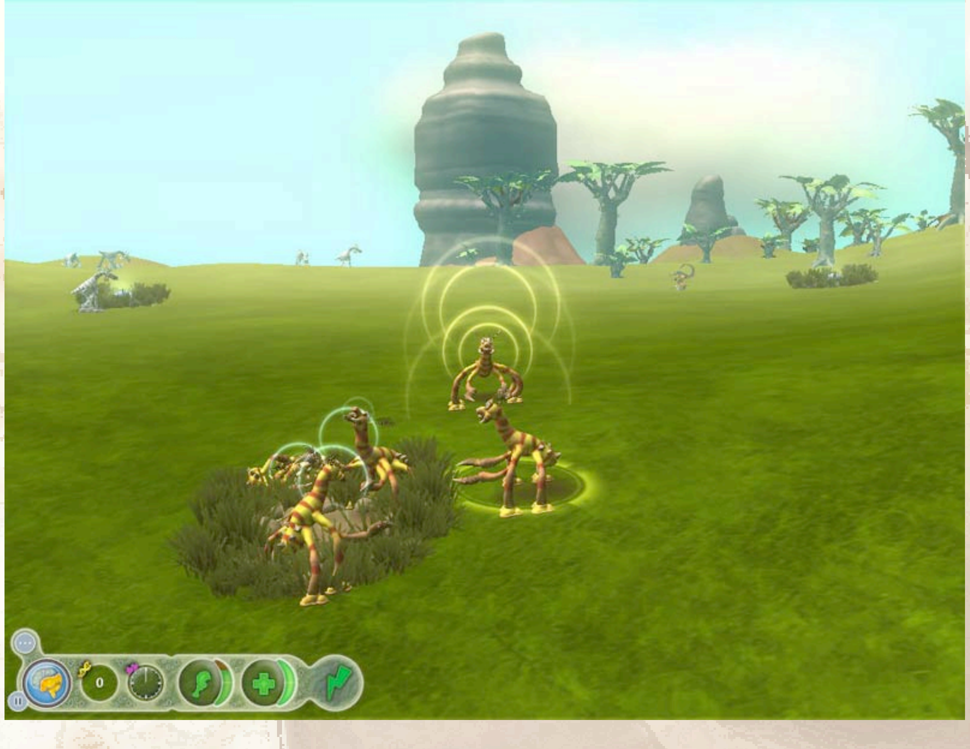
AlphaTestReference: ZERO

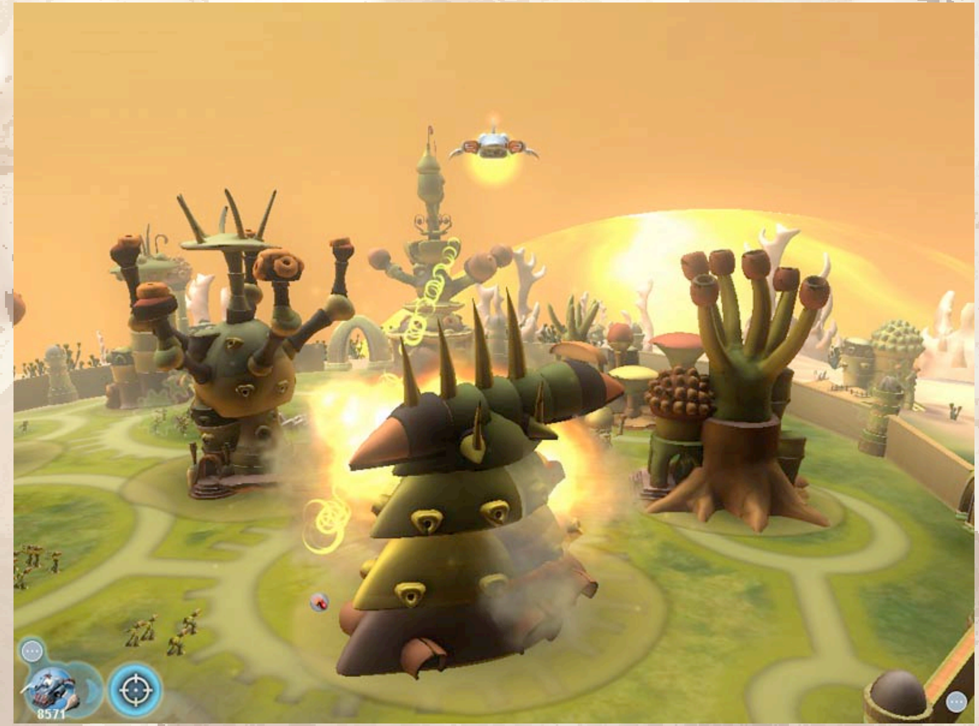


Paint & Proc UV Demo

Demo

Player-Created Models In-Game



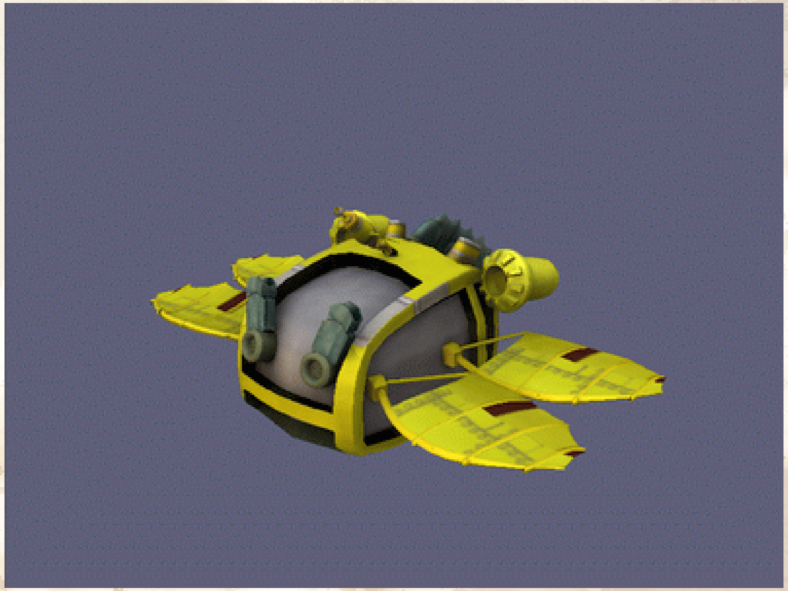


SPORE

Problems!

- Player-created model is not suitable for game use
  - Too many meshes (can be many parts)
  - Too many materials and textures
    (Parts x regions = a lot of batches)
  - Efficient rendering on GPU requires minimizing batch count.
  - No LOD!

Solutions



- Two solutions:
  - Texture Splatter
  - Mini Geometry Pipeline

Texture Splatter

- Generate unique UVs for ‘editor’ model as second uv set.

- Render model with *clipPosition = float4(uv2, 0, 1)*
  - *Splats* source textures into a single texture sheet
  - Allows resampling of high-res editor textures into a known-size texture, constant for all models
  - Old trick from Sims 2 neighbourhood lots

Unique UVs: Charting

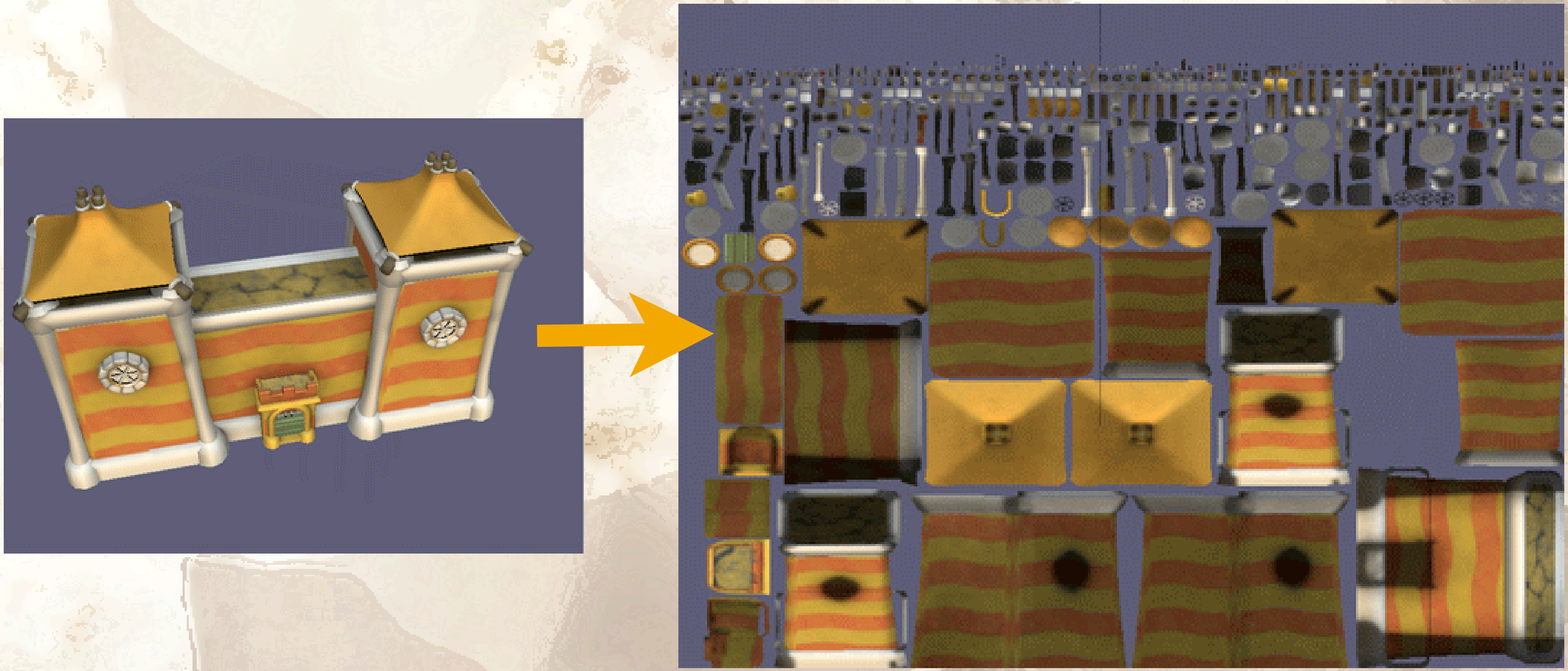
- Use face clustering to generate charts quickly
  - Generates ‘flattish’ chunks of geometry for charting
- Optionally run LSCM relaxation on these, otherwise planar project
- Use horizon-map-style packer (similar to Levy et al.)

SPORE

Ambient Occlusion!

- Problem with approaches that don't involve an entire-skin texture: no dark map
- We generate as a post-pass using GPU (accumulate shadow passes for model)
- Visual glue that holds everything together

Splatter + AO + Chart



DXT

- Standard DXT Compression is unusably slow
  - Brute force search for best two colours in 4x4 block
- Two ‘fast’ variants: id, internal EADXT lib
  - Both basically skip the search
  - Resulting colour blockiness too objectionable (downsampled version looks better)
  - Went with compromise: sample small number of points in search space and find best
  - Eliminated blockiness, 20ms on average machine for 512x512

Geometry Pipeline

- Can now weld all meshes together
- Generate LODs by **Vertex Decimation**
  - Faster than edge-based simplification: single pass
  - < 100 ms for 4 LODs, 50,000 vertices
- Has to handle vertex element discontinuities
  - Due to uvs, normals, tangents, bone weights
  - Must minimize discontinuities in output
  - While removing degenerate triangles, build chains of "shareable" vertices
  - Single-edge degenerate tris are key

Geometry Pipeline

- LOD Generation uses per-type LOD control settings:
  - Set simplification factor
  - Strip tangent space data
  - Force one weight per bone
  - Force the mesh to be static (i.e., strip all animation data)
  - Force mesh to share all vertices
  - DXT compression settings

Baking Details

- Desirable that we can do this while game is running
  - No blocking!

- Various chunks written as background jobs, controlled by job manager
  - Background load all source assets
  - Bake, then cache results to disk
  - Graphics assets need to be created in main thread

Creating Spherical Worlds

SPORE

James Grieve, David Lee Swenson, Andrew Willmott, Henry Goffin

Kate Compton, Ocean Quigley, Christian Stratton

Alumni: Ed Goldman, Eric Todd

Background

- Spore based on "powers of 10"
  - Cell life (2D world)
  - Planet: creatures, tribes, civilisations
  - Solar System
  - Interstellar
  - Galaxy

- Want seamless transitions

  → planets need to be **spherical**

Planet Constraints

- Need to have lots (millions? billions?)
  - many more than we can manually author
- Need to be playable
- Must look good
- Need to be fast to generate
  - We can’t store all these planets
  - Would like to transmit them at some point
- Need to support terraforming
  - Player modification of planet to support life

SPORE

Areas of Interest

- Parameterization
  - How do we store planet representation over surface? How do we store game data?
- Generating Heightfields
  - What are the operations? How can we make it fast?
- Texturing
  - Must be heightfield driven
- Authoring
  - Variety, art control

Parameterization

- Possible approaches:
  - Longitude/latitude (pole cap)
  - Gnomic
  - Freeform 3D: Sparse Voxel
  - Charts
    - Regular: cubemap, diamond, duodecahedron ...
    - On-the-fly (Voronoi-style)
    - Orthographic projection
    - Perspective projection

Parameterization Goals

- Minimize distortion and discontinuities
- Efficient (heightfield) storage
- Fast mapping from *(x,y,z)* to *(u,v)* and back

- Wrapping between charts
- Rectangular area splatting
- Efficient normal map generation

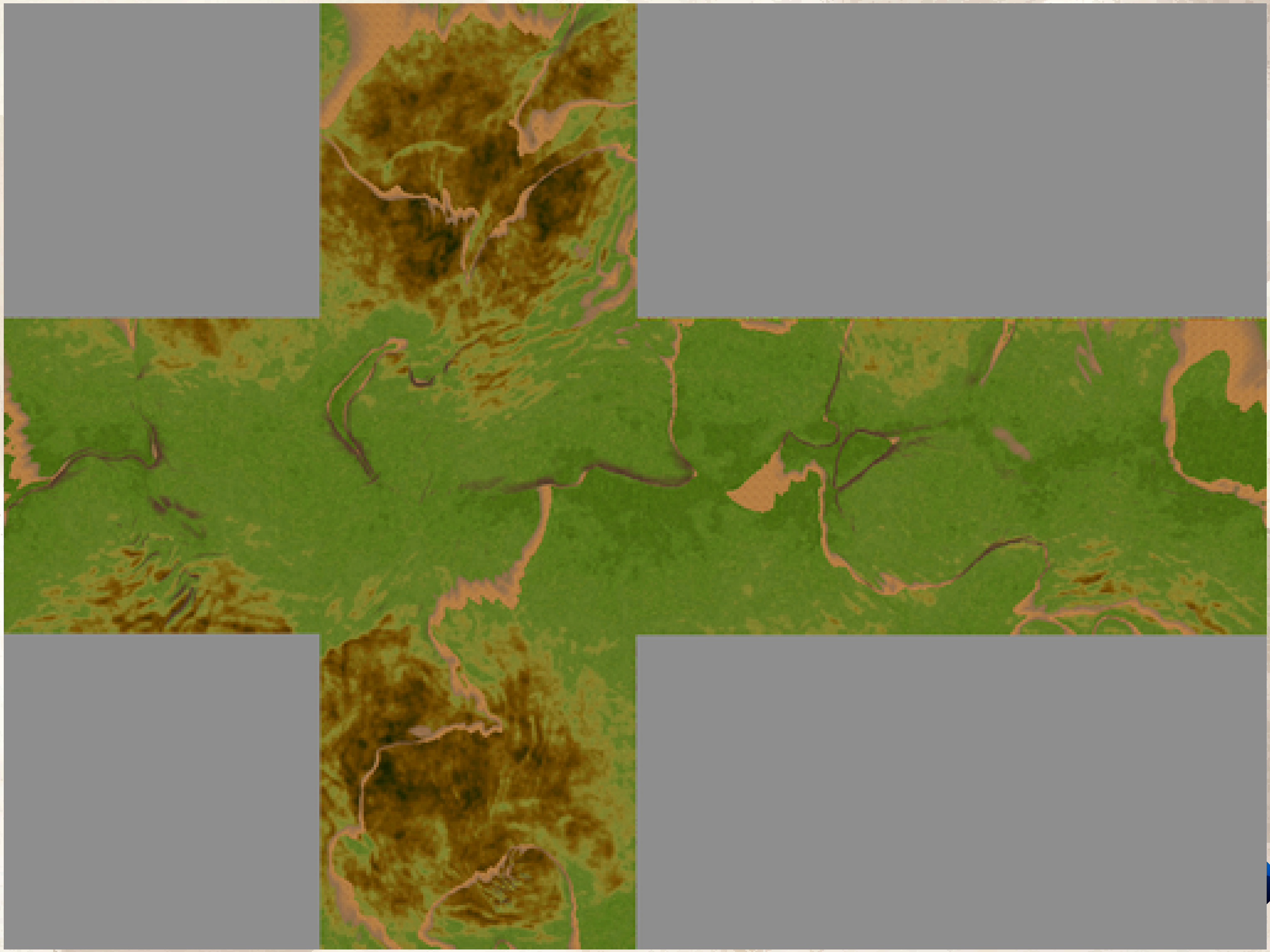
Parameterization: Cube Maps

- Chose cube maps as the best compromise

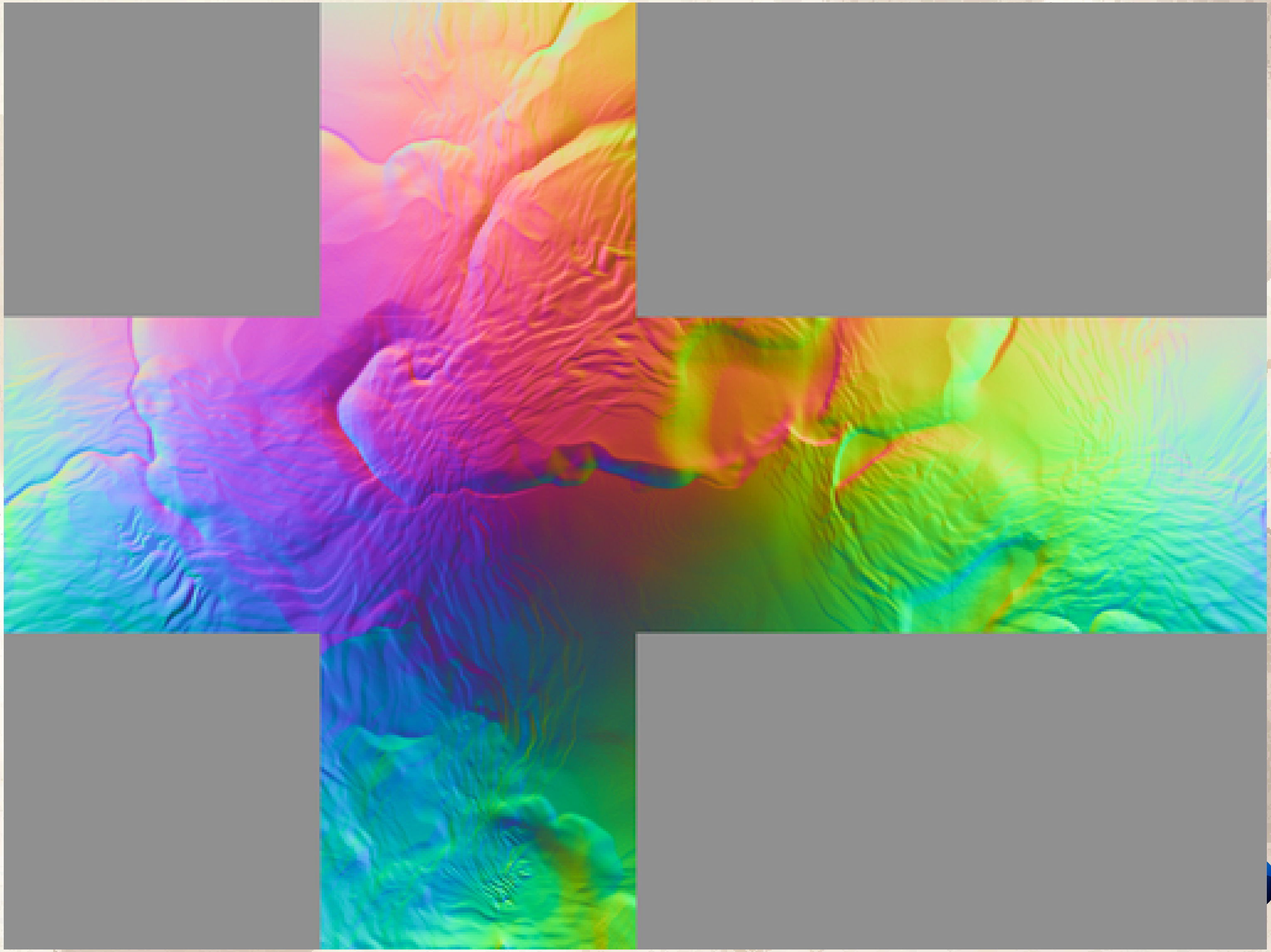
Parameterization: Cube Maps

- Chose cube maps as the best compromise
- Faces are grids
  - Familiar from previous games
- Distortion at corners
  - But not too bad, much better than pole distortion
- Face wrapping is tractable
  - Pick right face mappings -> simple permutation rules
- Projective projection
  - Lines map to great circles on sphere: very useful!

Colour Map



Normal Map



Normal Map

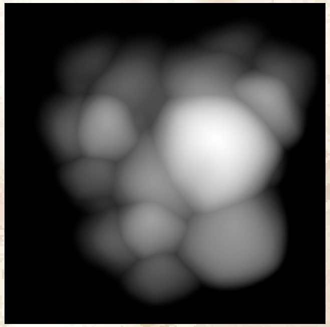
- Derived from height map
  - Large source of CPU time early on
- Standard DDF to find ‘flat’ normal map
  - Can then use Jacobian to warp to spherical form

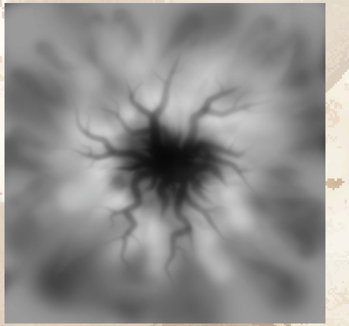
$$J(s,t,h) = \begin{pmatrix} h/w(1-s^2/w^2) & -sth/w^3 & -sh/w^3 \\ -sth/w^3 & h/w(1-t^2/w^2) & -th/w^3 \\ s/w & t/w & 1/w \end{pmatrix}$$

$$w = \sqrt(s^2 + t^2 + 1)$$

Generating Height Fields

- Brush system that operates on the sphere
- Brushes are 2D textured rects





- Several different brush operations
  - Conditionally raise or lower terrain
- Applied on GPU, after clipping brush footprint to faces

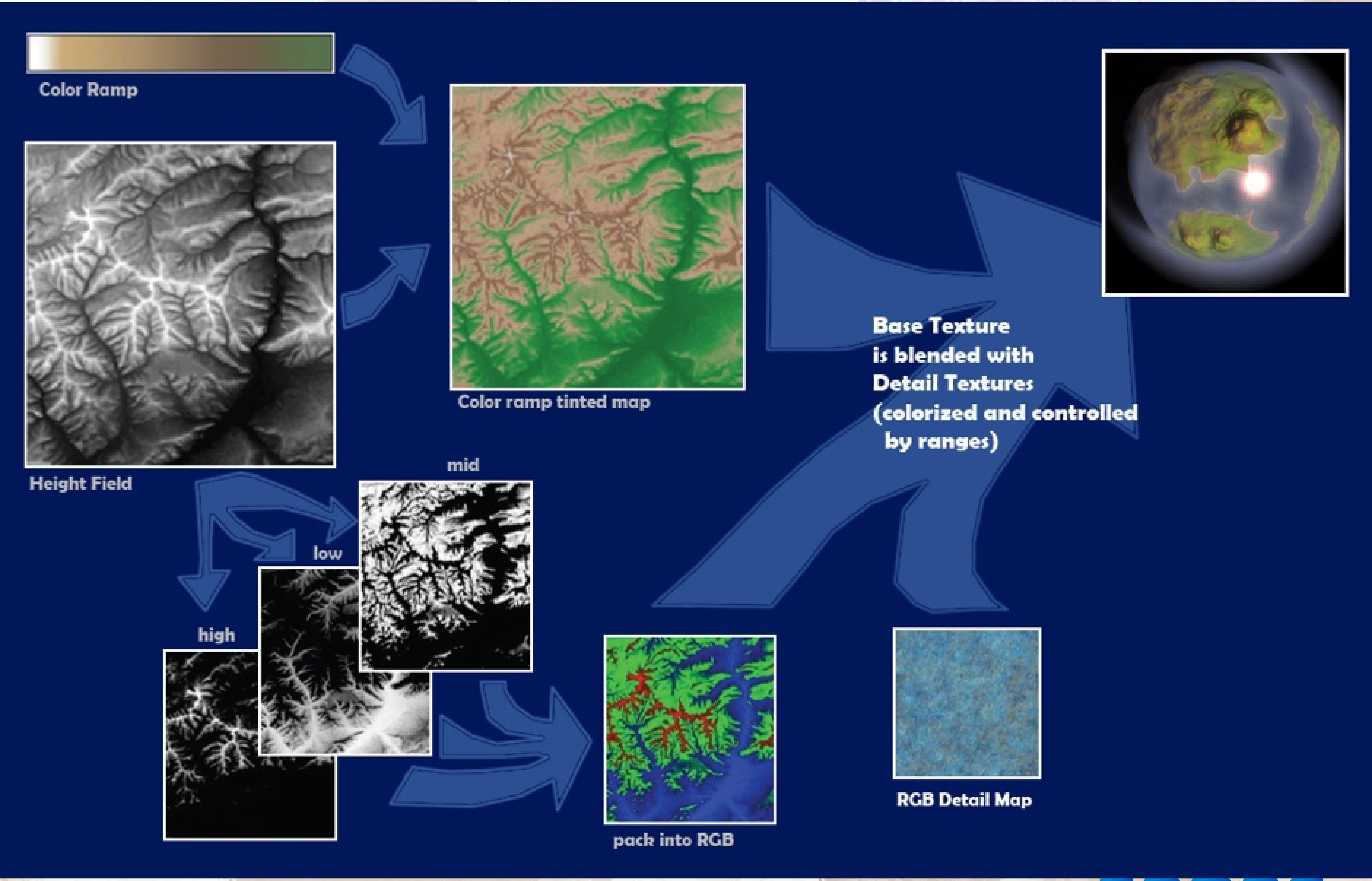
Controlling Terrain Brushes

- Use our effects system, Swarm, to run brushes over the surface

- Controlled by:
  - Particle systems (spawning other particle systems)
  - Randomized parameter ranges, random walks
  - Terrain forces
  - Force/control operates in the tangent plane

Texturing

- Derive Control Map from height field
  - Filter: water level, gradient, curvature
  - Combine according to tech artist formula
- Blends source textures to form base colour
  - Blends detail maps on the fly
- Planets have type, atmosphere, temperature
  - Control colour ramps, and atmosphere/fogging



Color Ramp

Height Field

Color ramp tinted map

mid

low

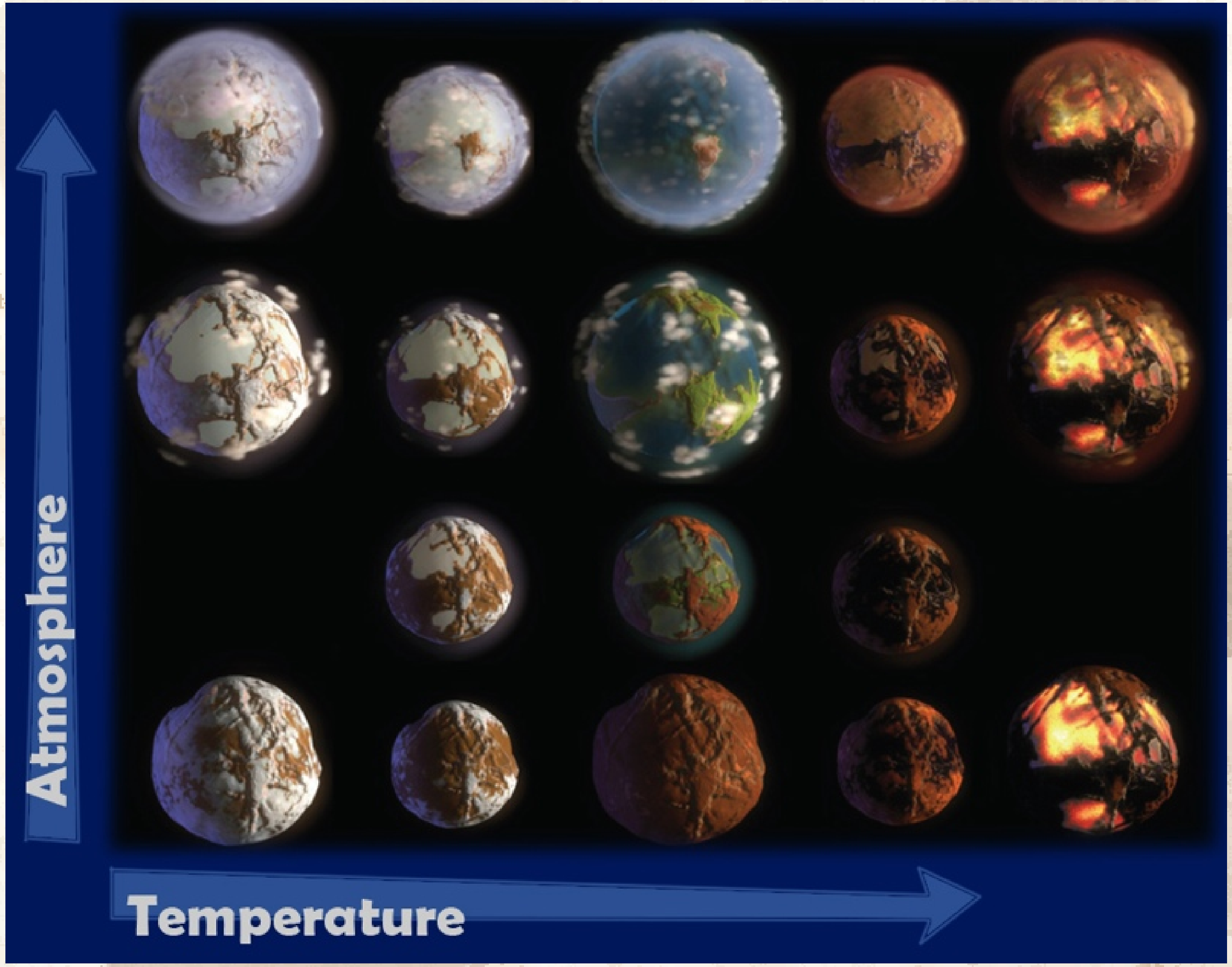
high

pack into RGB

RGB Detail Map

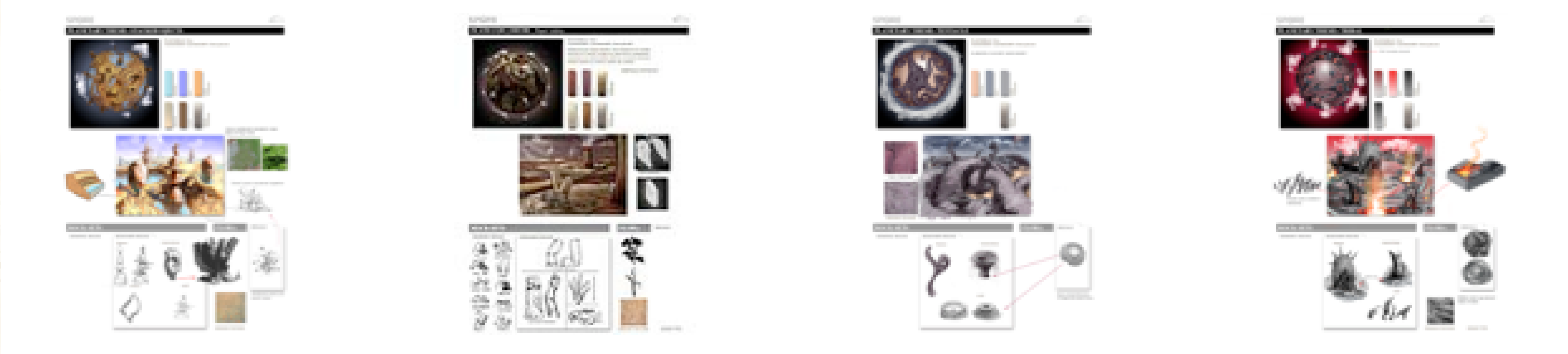
Base Texture
is blended with
Detail Textures
(colorized and controlled
by ranges)

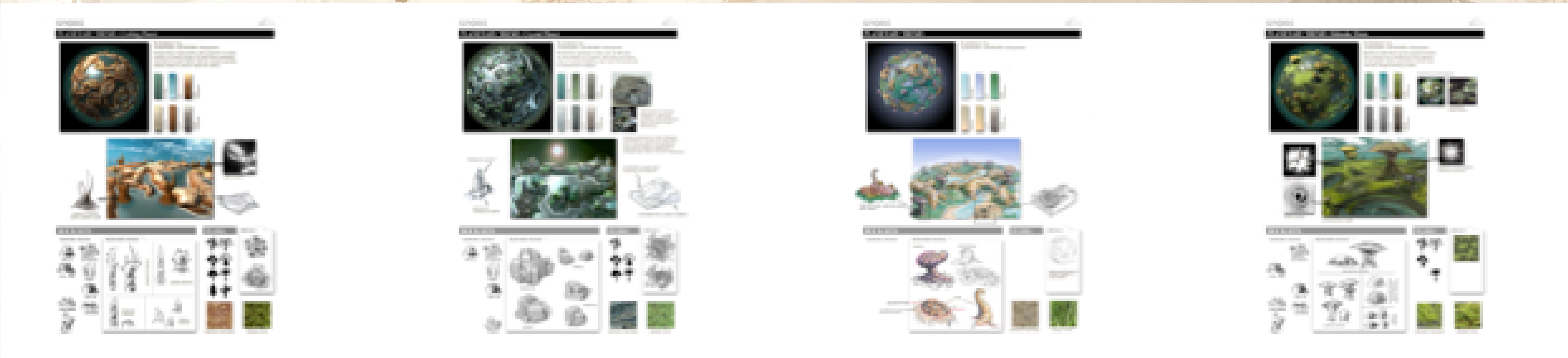
Terraforming

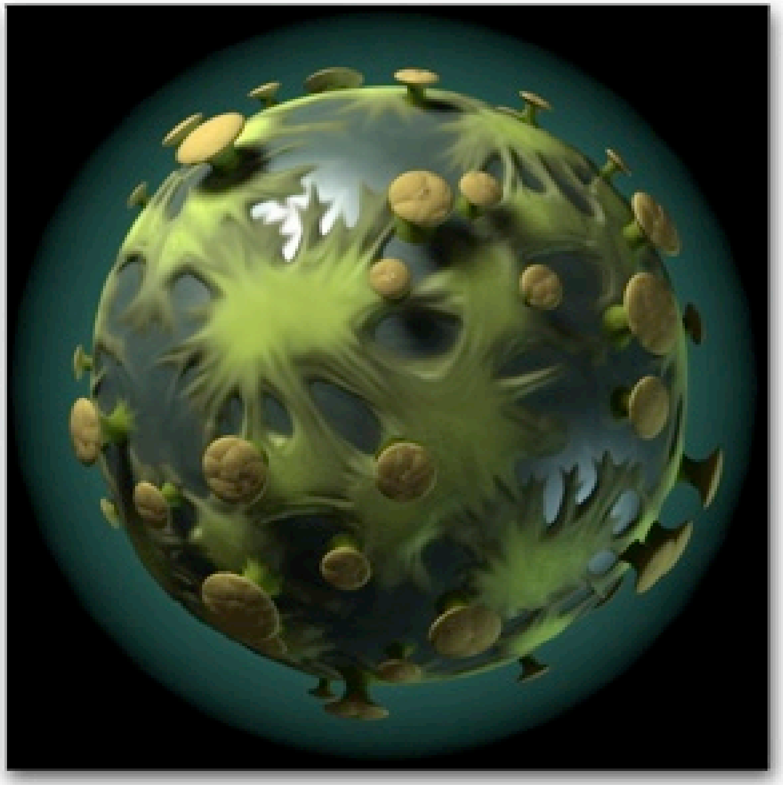


Authoring

- Concept Sketches







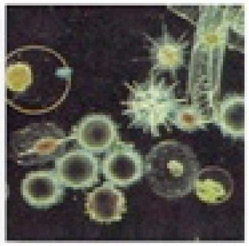
PLAYABLE: Yes
TAXONOMY CATAGORY: Storybook

Based on the floor of an ancient forest, this planet has landforms that appear to be ginat roots covered in moss and various fungii-looking rocks.

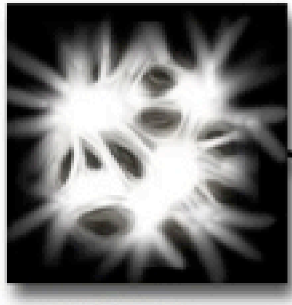
WATER
SKY
LIVE TERRAIN
BEACH
CLIFF
DEAD TERRAIN

PARTICLE EFFECTS

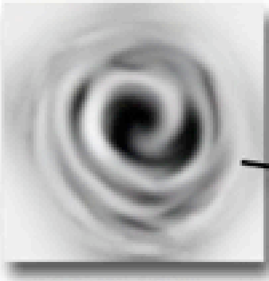
CLOUD PATTERNS



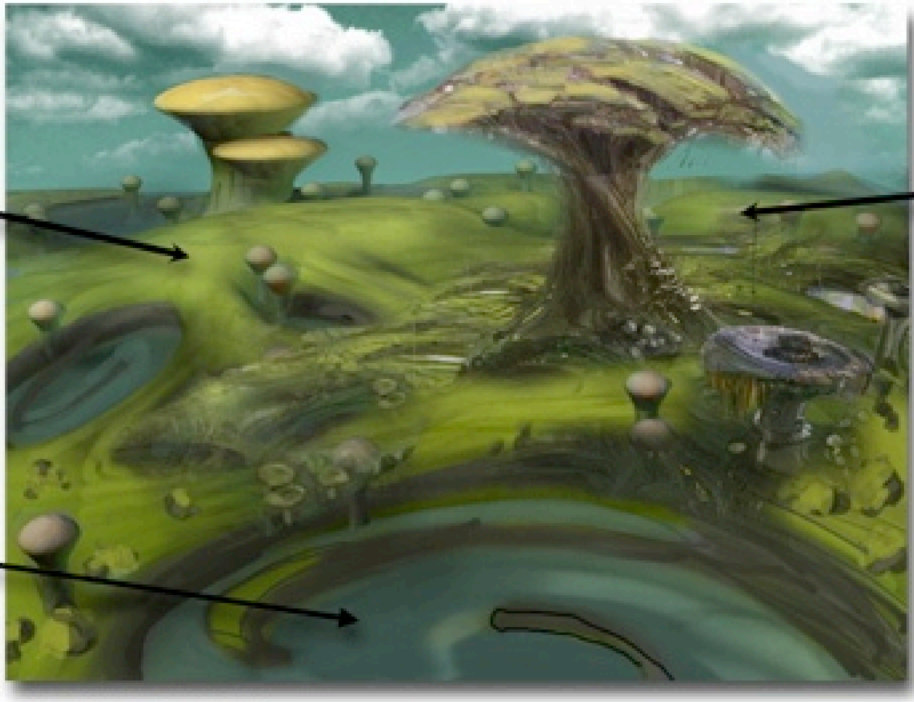
LOOPBOX PARTICLES



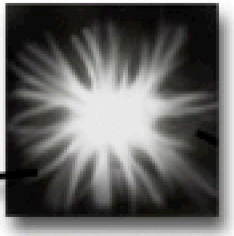
LAND SCRIPT



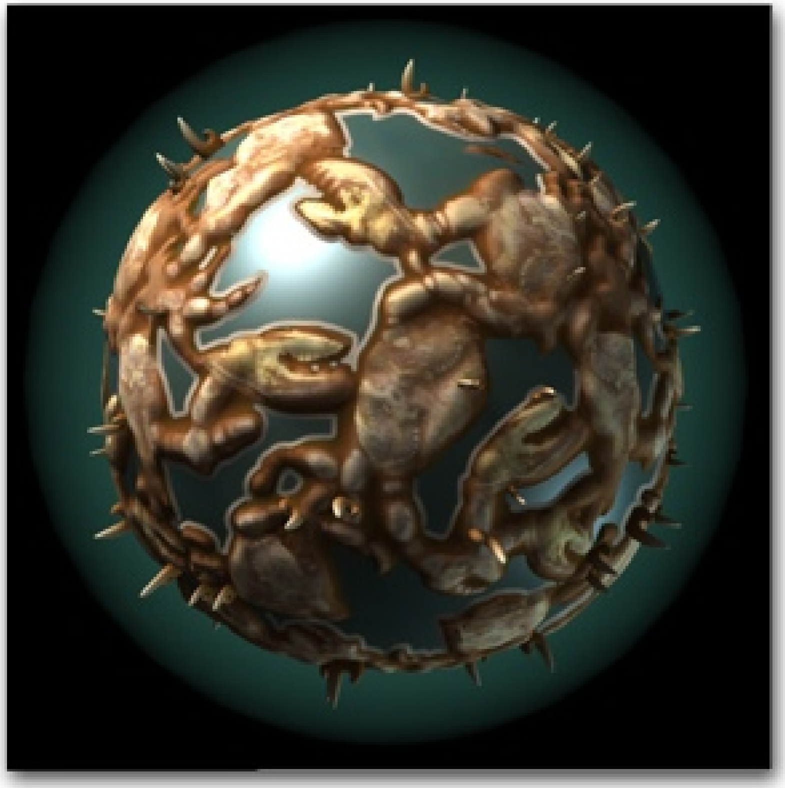
POND SCRIPT



CIVILIZATION VIEW

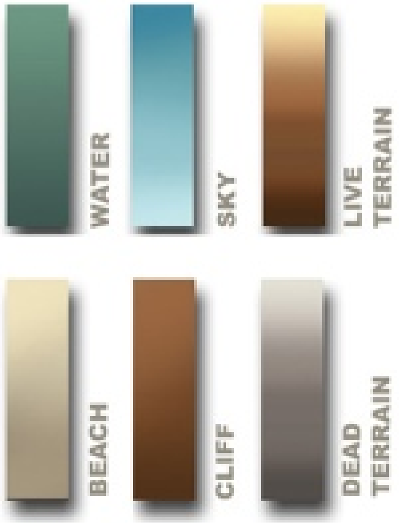


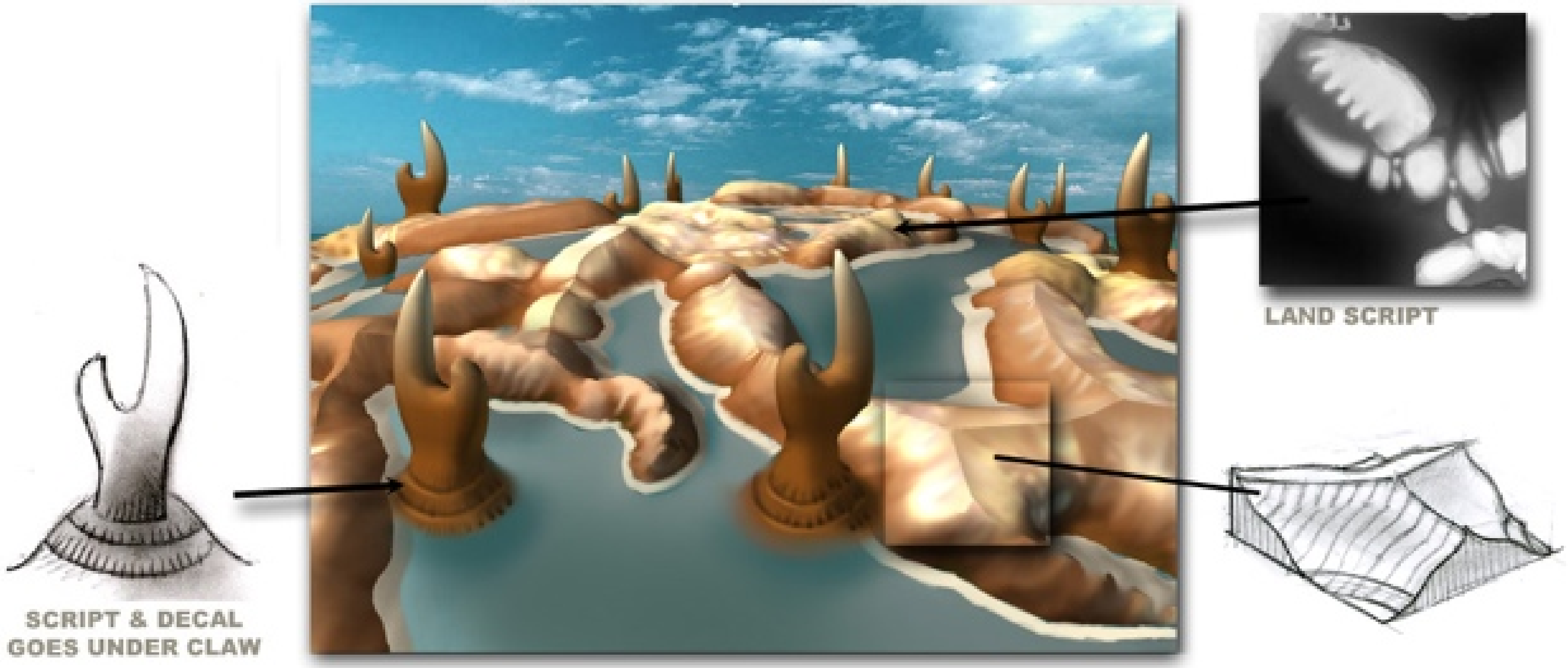
SCRIPT TO GO UNDER MASSIVE OBJECT



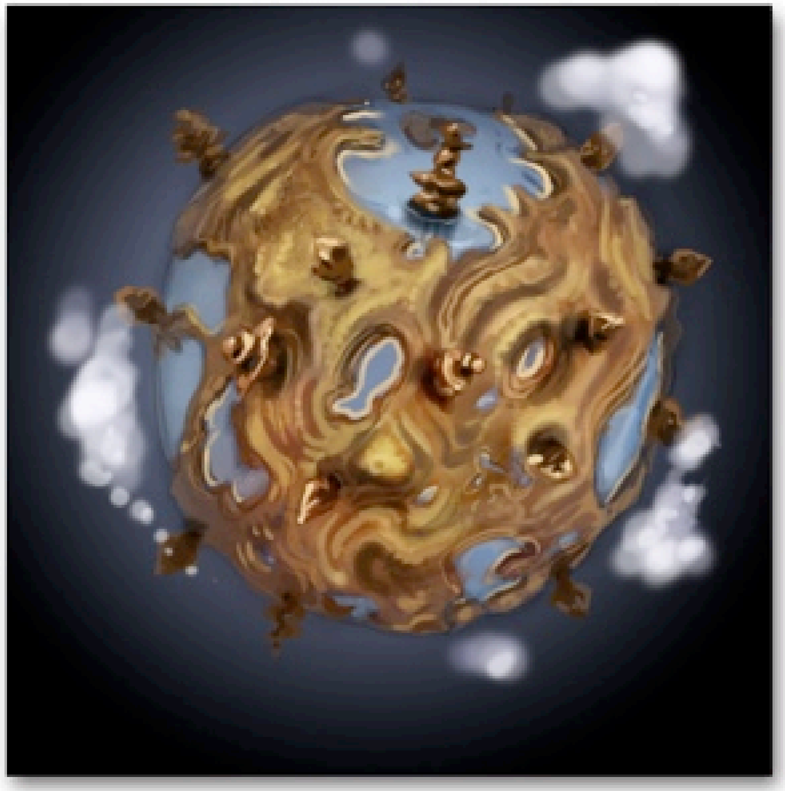
PLAYABLE: Yes
TAXONOMY CATAGORY: Storybook

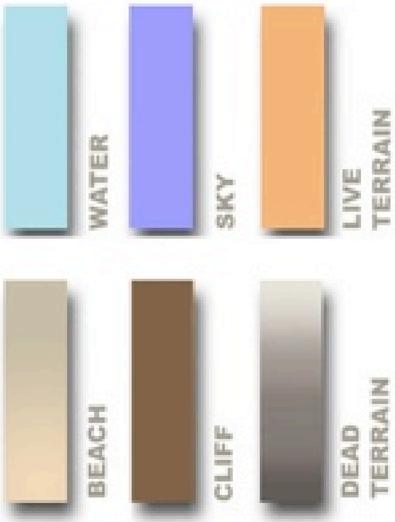
Inspired by crab shells, this planets is made mostly of small strips of land that randomly connect to each other and to a main section where there is more room for cities.

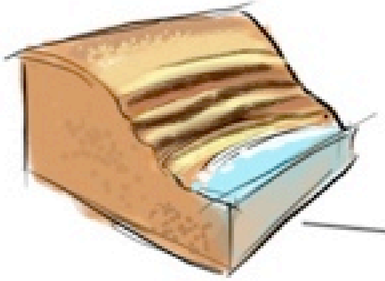


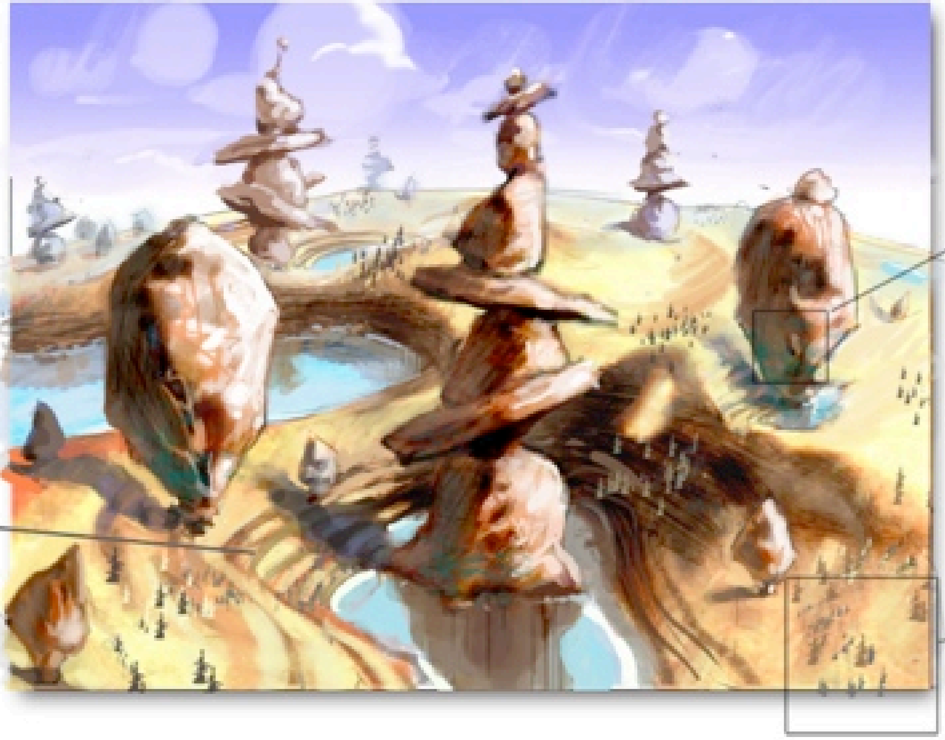


PLAYABLE: Yes
TAXONOMY CATAGORY: Storybook



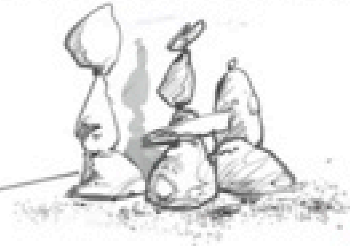






moss scattered randomly near base of big rocks

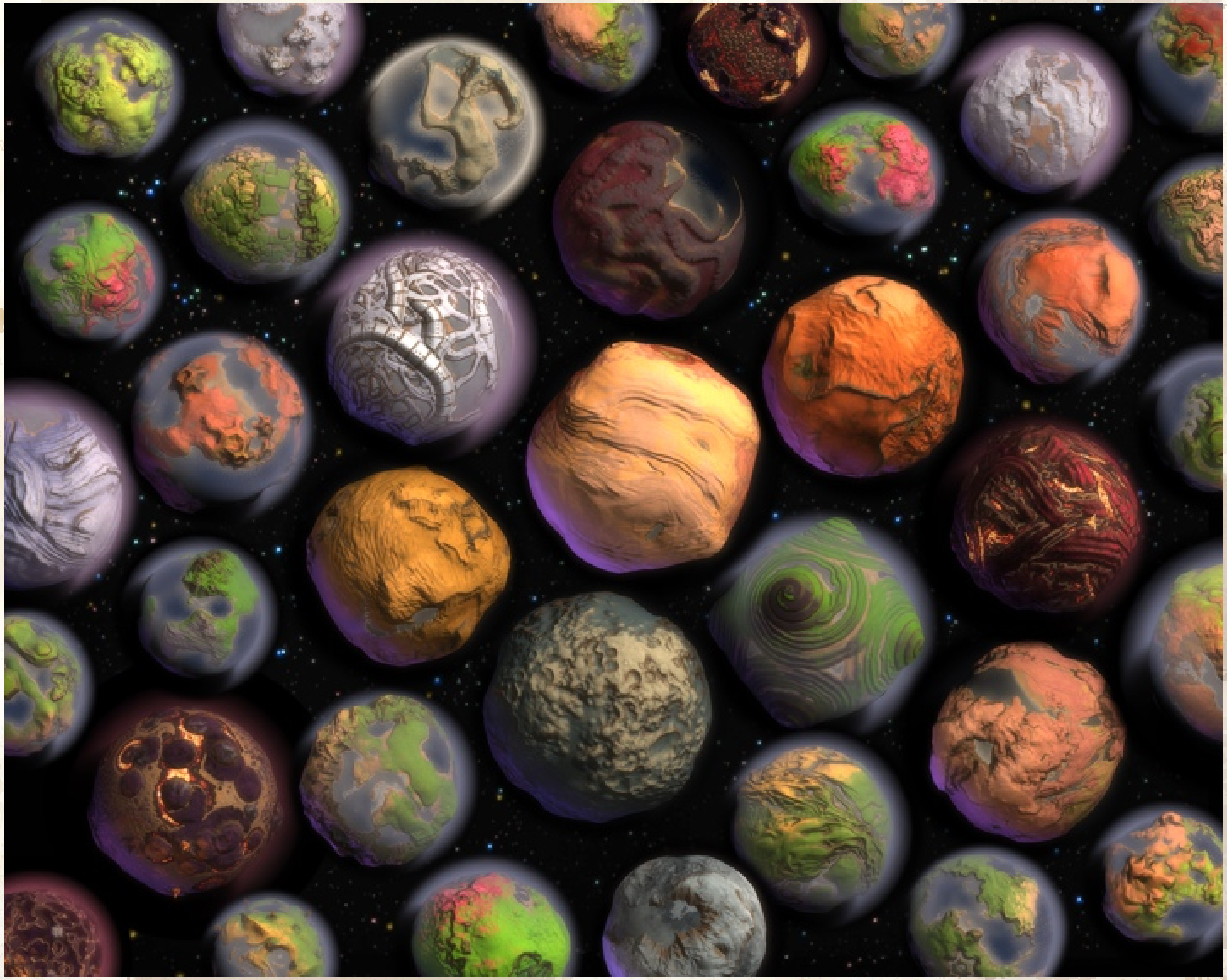
Small rocks clustered together



Authoring

- Originally one mega effects script
  - random selection between various child effects
- Difficult to control
  - Hard to get art-directed
- Introduced a top layer with more control: *terrain scripts*
- Each script produces a particular kind of planet

The Result



Authoring: Planet Editor

Demo

SPORE

Summary

- Do for content what MMO's have been doing for gameplay
- Lot of tech!
- High-risk
- Pre-pro helped reduce risk, but still a lot during production
- But we hope it's worth it

Questions?